

AD-A103 174

SRI INTERNATIONAL MENLO PARK CA
AUTOMATIC PALMPRINT VERIFICATION STUDY.(U)
JUN 81 J R YOUNG, R W HAMMON

F/G 9/2

F30602-79-C-0207

UNCLASSIFIED

RADC-TR-81-161

NL

1 OF 1
AD-A
103 174

END
DATE
FILMED
10-81
DTIC

AD A103174

LEVEL

12

RADC-TR-81-161
Final Technical Report
June 1981



AUTOMATIC PALMPRINT VERIFICATION STUDY

SRI International

James R. Young
Robert W. Hammon



APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

DTIC FILE COPY

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

81 8 21 058

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-81-161 has been reviewed and is approved for publication.

APPROVED:

John V. Ferrante

JOHN V. FERRANTE, 1Lt, USAF
Project Engineer

APPROVED:

John N. Entzminger

JOHN N. ENTZMINGER
Technical Director
Intelligence and Reconnaissance Division

FOR THE COMMANDER:

John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (IRAA) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

19 REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM	
1. REPORT NUMBER RADC-TR-81-161	2. GOVT ACCESSION NO. AD-A103 174	3. RECIPIENT CATALOG NUMBER	
4. TITLE (and Subtitle) AUTOMATIC PALMPRINT VERIFICATION STUDY.		5. TYPE OF REPORT & PERIOD COVERED Final Technical Report, Aug 79—Jan 81	
6. AUTHOR(S) J. R. Young R. W. Hammon		6. PERFORMING ORG. REPORT NUMBER SRI Project 8788	
7. PERFORMING ORGANIZATION NAME AND ADDRESS SRI International 333 Ravenswood Menlo Park CA 94025		8. CONTRACT OR GRANT NUMBER(s) F30602-79-C-0207/	
9. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (IRAA) Griffiss AFB NY 13441		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62710H DNA0211	
11. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Same		12. REPORT DATE June 1981	
		13. NUMBER OF PAGES 94	
		14. SECURITY CLASS. (of this report) UNCLASSIFIED	
		15. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.			
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) Same			
18. SUPPLEMENTARY NOTES RADC Project Engineer: John V. Ferrante, 1Lt, USAF (IRAA)			
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Optical Processing Statistical Analysis Access Control Feature Extraction Data Reduction			
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) A computer-based system has been created using 29 features of hand geometry to discriminate successfully 278 images of hands taken from 30 male and female subjects over a four-month period. These features include hand perimeter and area measures, finger lengths and widths, palm widths, and certain ratios of lengths and widths. A linear discriminant procedure was used to classify all images according to the subjects from which the images were taken. All images were correctly			

DD FORM 1473

JAN 73

EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED


SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

414201

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

classified for 100% results.



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

CONTENTS

LIST OF ILLUSTRATIONS	v
ACKNOWLEDGMENT	vii
I INTRODUCTION	1
II DATA COLLECTION	3
III DATA PREPARATION	7
A. Data Generation	7
B. Data Reduction	8
IV DATA ANALYSIS	9
V CLASSIFICATION RESULTS	13
VI COMPUTER PROGRAM SUMMARIES	19
A. HIST	19
B. THRESH	20
C. LINEBYLINE	20
D. SCAN	20
APPENDICES	
A--COMPUTER PROGRAM LISTINGS	A-1
B--DOCUMENTATION OF LINEBYLINE	B-1
C--ANNOTATIONS FOR THE SEPARATE SPSS RESULT PACKAGE (Submitted Separately)	C-1

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By _____	
Distribution/ _____	
Availability Codes	
Avail and/or	
Dist	Special
A	

ILLUSTRATIONS

1	Apparatus Used for Data Collection	4
2	Fingertip Location Measurement	10
3	Measurement Points for Lengths of Finger and Thumb	10
4	Finger and Thumb Width Measurement Points	11
5	Curvature and Distance Measures	12
6	Analysis of Hand Geometry Features	14
7	Analysis of Hand Geometry Features Using Within-Groups Correlation Matrix	16

ACKNOWLEDGMENT

With appreciation, the authors gratefully acknowledge the substantial contributions of the following toward the research reported in this document:

- Dr. John Ostrem served as a consultant on feature-extraction methods, on data analysis, and--particularly--on the use of commercial software used in the statistical analysis of the feature-extraction results.
- Mr. Gregory Myers supplied the computer program LINEBYLINE and provided guidance in its application to this research. He also gave valuable help and advice in feature selection and analysis.
- Mr. Zev Pressman gave valuable guidance in the design of the data-collection apparatus and in its initial adjustment and evaluation.
- Mr. Marshall Wilson recruited subjects, photographed the subjects' hands, coordinated the film processing, and cataloged all of the data in the data base.

I INTRODUCTION

This report describes a one-year research and development effort to determine the feasibility of using hand geometry and other features of the hand as bases for the construction of an automatic personnel authentication system for access control. The subsequent sections of the report describe the method and extent of data collection, the method of data preparation and reduction, the data analysis, the experimental authentication results, and provide brief descriptions of the computer programs that were written for this work.

The appendices are photocopies of listings of the computer programs used and of critical parts of the experimental statistical analysis and raw data.

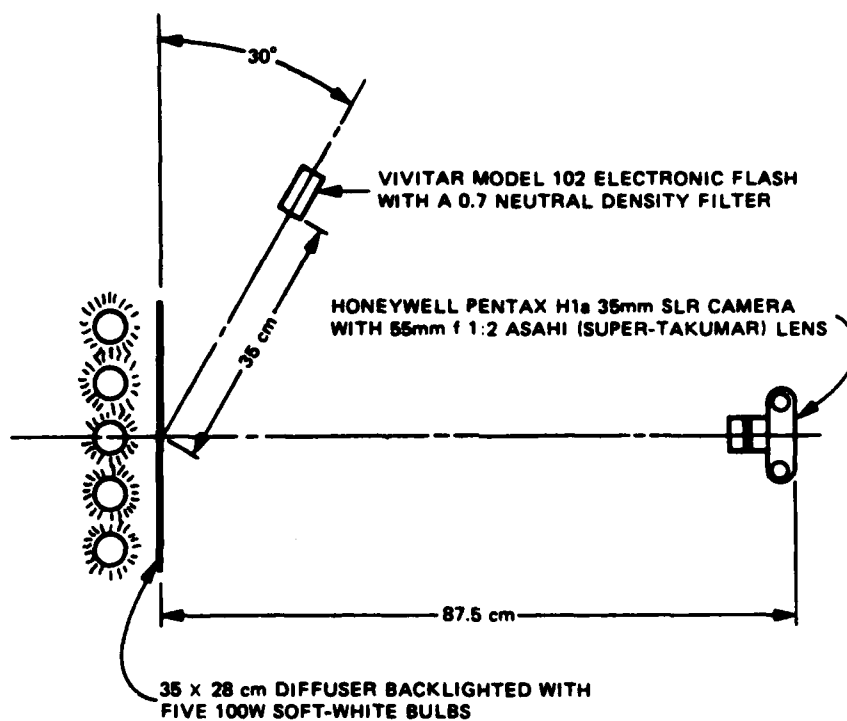
II DATA COLLECTION

Previous access-control systems using hand geometry or fingerprints have used devices that capture data from physical contact with or proximity to the hand surfaces being measured. Such contact or closeness can distort the measurements, particularly if the hand can move or exert pressure. A measurement system that can avoid distorting the features being measured will be more reliable, which in turn will reduce the occurrence of incorrect authentication decisions.

Precise orientation of the hand in space and precise control of finger and thumb spread are difficult to accomplish without some mechanical device, however. To compensate for the variable position of the hand from trial to trial (because there was no device to hold it in a fixed position) it was decided to concentrate on adjusting the data. Precise data reduction would allow easier and more natural interaction between the person seeking authentication and the access-control system.

The system used to collect data is schematically illustrated in Figure 1. The subject's hand is positioned near the light diffuser; the open palm faces the camera and the fingers are comfortably spread. The subject's name and session number are displayed in the upper quarter of the imaged area and thus become an integral part of the data base.

The data base collected contains 50 subjects, 25 males and 25 females, ranging in age from the early 20s to 65, in height from approximately 5' to 6'3", and in weight over a normally observed distribution. The total number of sessions (five trials per session) was 1020, approximately evenly divided between males and females. Of these, 997 sessions produced 4985 high-quality images with full gray scale. In the remaining sessions, flash problems of failure to fire or synchronization incurred; the resultant images were quite useful for geometric measurements but contain no gray-scale features.



APERTURE = $f/16$
 SHUTTER SPEED = $1/30$ s
 FILM: KODAK HIGH-CONTRAST
 COPY FILM 8069
 OBJECT MAGNIFICATION: 0.07

FIGURE 1 APPARATUS USED FOR DATA COLLECTION

Negative images have been cataloged and stored in three volumes that were designed and manufactured for such use. Photo development followed standard procedures for the black-and-white high-contrast copy film (Kodak 5069). The film manufacturer states that if storage conditions are correct (cool and dry) the archival qualities of this film are excellent--it will retain quality for more than 30 years.

Image quality is also excellent. Correctly focused and exposed negatives have been examined with a 400X microscope, and minutiae are clearly visible on fingertips.

III DATA PREPARATION

A. Data Generation

Two methods were used to obtain digital-image data to be used for further processing.

- An SRI-built Reticon CCD array scanner (512 elements per line) interfaced to an LSI-11 was used early in the project.
- A commercially available Eikonix digitizer (1024-element Reticon \times 1500 scan steps) was used to supply final evaluation data.

Data from the first system were generally unsatisfactory because of a combination of problems. The system interface included a hardware thresholder for the analog data generated by the Reticon array. The user can vary the threshold, but it is uniform for all samples coming from the CCD array. Such an arrangement works well only if the image area is uniformly lighted, the individual CCD elements are reasonably uniform in sensitivity, and the scanned images have equivalent niecom densities. Unfortunately, these conditions were not satisfied, and the uniform hardware threshold was found to be quite unsatisfactory. It is probably possible to build a system that utilizes a fixed-level threshold for this type of application; however, such a device with a sufficiently uniform large field of view was not available, and the uniform-threshold technique has therefore not been thoroughly evaluated.

The Eikonix digitizer is an excellent instrument for purposes of this study. The digitized images generated are 1024 \times 1500-element data each with 8 bits of gray scale recorded on magnetic tape. The real resolution of the data when the film negatives are sampled with this spatial frequency is two pixels (picture elements) per mm. The system can automatically compensate for lighting nonuniformities and variable CCD-element sensitivities. It is also fairly fast: one complete cycle of generating and recording an image requires about one minute. All of the data used to generate the results appearing in this report were collected through the Eikonix system.

B. Data Reduction

Image data on magnetic tapes were transported to a PDP 11/40 computer system in SRI's Bioengineering Research Center. Each image was reduced by extracting the pertinent 512×512 -element field and by thresholding to a binary image. Finally the binary image was reduced to a hand-perimeter list of X-Y coordinates that was stored on a DEC RK 05 disk. Although this process is somewhat involved, it does provide final data that are quite reliable and easy to work with in subsequent computations. (It is to be noted that the final data set generated by this procedure does not include any information about a subject's hand other than the perimeter coordinates in a 512×512 -element field.)

A very important feature of the data-reduction process is the method used to compute the perimeter list from a 512×512 -element hand image. The method involves using a computer program (LINEBYLINE) that was developed earlier at SRI and recently rewritten in PASCAL for use on the PDP 11/40. The program generates the perimeter list while accessing the image data one line at a time, starting at one edge of the field. This implies, for a practical system, that a minimum of memory and data buffering is required; consequently, there is a significant saving of money and execution time.

IV DATA ANALYSIS

The basis for the analysis and the geometric features that have been extracted is, in each case, a list of the X-Y coordinates of the hand perimeter. Such a perimeter is described with a real resolution of two pixels per mm.

All measurements are based on landmarks that are easily identified on the perimeter; the tips of the fingers and thumb were selected for this study. These landmarks were identified by calculating a coarse curvature function over the perimeter and finding criterion shifts in the angle of a tangent to the perimeter. Tangents were estimated by passing an 18-point window over the perimeter and using the window end points to define a vector in the X-Y image plane. The window was moved in nine-point increments. This analysis gives a rough estimate of the tip location.

The rough estimate is refined by locating four additional points on the finger, two found 25 mm away along the perimeter on each side of the estimated tip and two more found 35 mm away on each side. The virtual lines connecting each pair of points are bisected, and the intersection of the tip and the line defined by the bisection points is defined to be the actual tip location. This process is illustrated in Figure 2. After a precise location for each tip is established, the following features can be extracted:

- Features 1 through 5 are lengths of the fingers and thumb, starting with the little finger. Lengths are measured from the tip to the depth of the interdigital space as illustrated in Figure 3.
- Features 6 through 10 are widths of the fingers and thumb. A width is the distance between two points on the perimeter of a finger; the points are 45 mm down the perimeter on each side of the tips of the little finger and the thumb, and 60 mm on each side of the other fingertips. Width measurement points are illustrated in Figure 4. These features are ordered as were the first five.

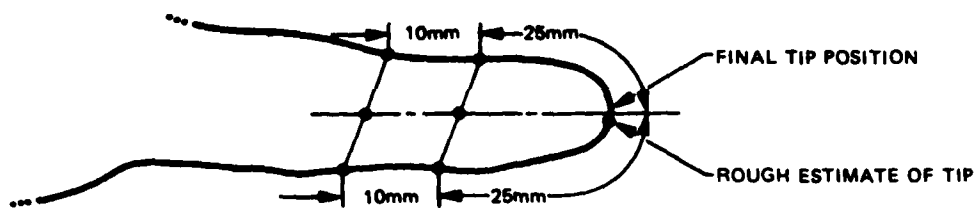


FIGURE 2 FINGERTIP LOCATION MEASUREMENT

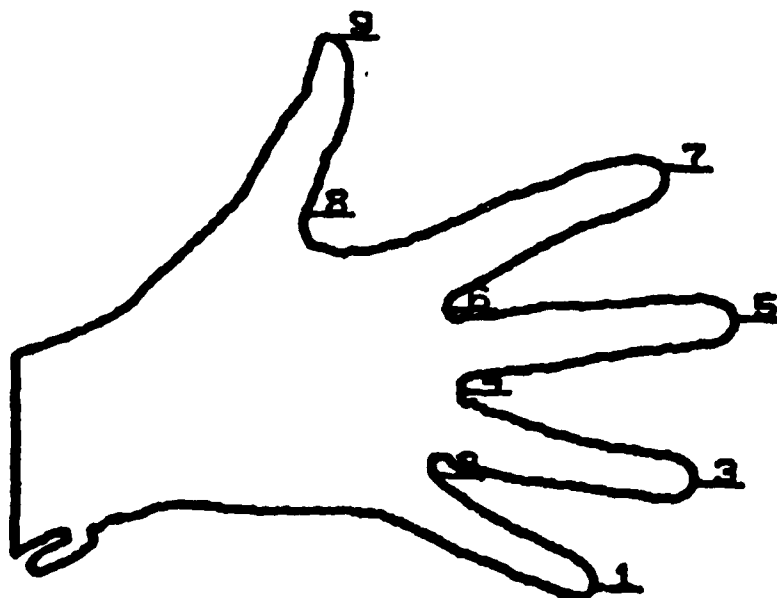


FIGURE 3 MEASUREMENT POINTS FOR LENGTHS OF FINGERS AND THUMB

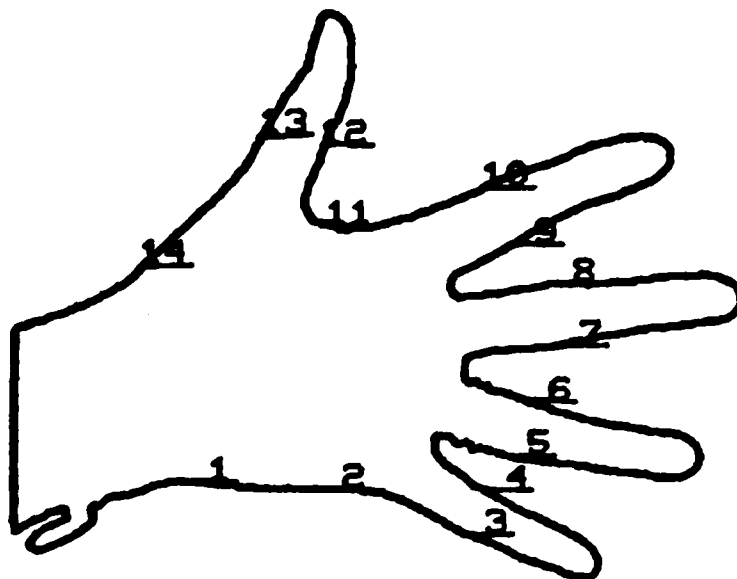


FIGURE 4 FINGER AND THUMB WIDTH MEASUREMENT POINTS

- Features 11 through 15 are ratios of the measurements of finger lengths to widths.
- Feature 16 is the hand width measured between two points on the palm. One point is 9 cm on the perimeter from the tip of the little finger toward the wrist and the other is 11 cm on the perimeter from the tip of the index finger toward the thumb. These points are labeled 2 and 11, respectively, in Figure 4.
- Features 17, 18, and 19 are the hand perimeter, hand area, and the ratio of the squared-perimeter length to the area. The perimeter and area are measured from point 1 to point 14 in Figure 4. Point 1 is 13 cm on the perimeter from the tip of the little finger, and point 14 is 10 cm on the perimeter from the tip of the thumb.
- Ten features, 20 through 29, are paired values corresponding to the curvature of each fingertip and the distance over which the tip shape can be represented by a circular arc with less than a criterion value of rms (root mean square) error. The process of extracting a typical feature pair is illustrated in Figure 5. A tangent to the perimeter, such as T, has an angle, α , as shown. If α is plotted as a function of distance along the perimeter defining a finger, then a plot such as the one shown in Figure 5 results. The slope of the plot between P_1 and P_2 is related to the curvature of the tip. The slope is recorded as the curvature. A correlation calculation over the region containing this characteristic region yields a high value

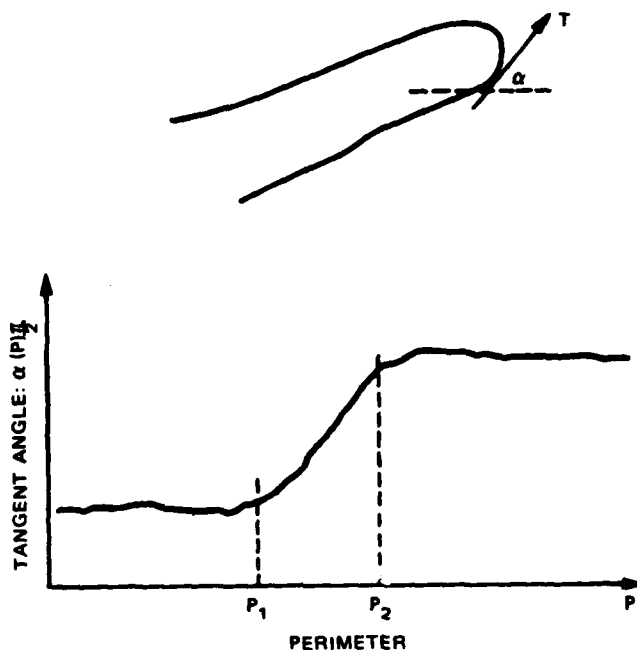


FIGURE 5 CURVATURE AND DISTANCE MEASURES

until the boundary of the region extends into the flat segments on either side of the sloping section. P_1 and P_2 are the boundary points when the correlation coefficient (using a straight-line fit to the sloping region) falls to 0.98. The distance between P_1 and P_2 is recorded as the perimeter length over which the fingertip can be approximated well with a circular arc.

In summary, hand geometry is expressed as a set of 29 features that are measured by locating precisely five landmarks on the perimeter of the hand--the tips of the fingers and the thumb. The feature set contains 16 measures of lengths and widths and ratios of lengths and widths, and three measures of area or area and perimeter length. Ten measures of shape (curvature) are included.

V CLASSIFICATION RESULTS

Thirty subjects (15 male, 15 female) were selected randomly from the subject population and 278 images taken from the selected subjects were digitized. The data were converted to perimeter lists and sets of 29 features were extracted. The data were submitted to a standard statistical analysis program* to determine the power of the extracted features in discriminating among the 30 subjects. Linear discriminant analysis was performed and all 278 feature sets were correctly classified for all subjects. The detailed results of this analysis, as well as the feature data set, are submitted as a separate package. Annotation for this package is included as Appendix C.

Of critical importance are indicators of value and reliability of the various features selected. One such indicator is the univariate F-ratio, a number that expresses the ratio of between-groups variation of a given measure and the within-groups variation of that same measure. Large values of F indicate that a measure will be relatively useful in separating groups (or samples from different subjects) and small values predict that a measure will do poorly. Figure 6 summarizes the F-ratios of the selected 29 features over the 30-subject, 278-sample data base.

All of the features appear to be useful and reliable, especially the more global measures of hand width, perimeter, and area. Some measures, notably the curvature and distance measures associated with the thumb are marginal as presently calculated. They could be made more useful by improving the algorithm that deals with the curvature of the perimeter in the vicinity of the thumb and by locating the thumb tip more precisely.

* Statistical Package for the Social Sciences (SPSS), Version 7.0.

WILKS LAMBDA (U-STATISTIC) AND UNIVARIATE F-RATIO

VARIABLE	WILKS LAMBDA	F
FEATUR01	.0832	94.2725
FEATUR02	.1984	34.5555
FEATUR03	.0326	253.7253
FEATUR04	.0411	199.7288
FEATUR05	.1254	59.6265
FEATUR06	.0889	87.6057
FEATUR07	.0884	88.1511
FEATUR08	.0497	163.4109
FEATUR09	.0396	207.2642
FEATUR10	.3469	16.1010
FEATUR11	.1150	65.8000
FEATUR12	.2728	22.8001
FEATUR13	.0961	80.4311
FEATUR14	.0704	113.0063
FEATUR15	.3051	19.4798
FEATUR16	.0282	295.1276
FEATUR17	.0185	453.6141
FEATUR18	.0152	554.9787
FEATUR19	.0450	181.5804
FEATUR20	.4010	12.7763
FEATUR21	.1229	61.0282
FEATUR22	.1865	37.3090
FEATUR23	.2264	29.2203
FEATUR24	.1555	46.4599
FEATUR25	.1325	55.9899
FEATUR26	.4777	9.3499
FEATUR27	.2048	33.1988
FEATUR28	.7087	3.5142
FEATUR29	.5388	7.3202

FIGURE 6 ANALYSIS OF HAND GEOMETRY FEATURES

In general, it appears that the measures used are being generated very reliably, which results in good (high) F-ratios. One difficulty in previous systems that used hand geometry for authentication was unreliable measures of certain features, with correspondingly small F-ratios and poor discriminating power.

Figure 7 gives more information about the quality of the feature set. In this table, the within-groups correlation matrix is listed. The correlation matrix is used to express the degree to which pairs of features are correlated or dependent. Circled are the two largest magnitudes of correlation values, corresponding to the pairs

- Second finger length and the ratio of its length and width
- Thumb width and the ratio of its length and width.

It is to be expected that ratios of features and the features themselves will tend to have relatively large correlation magnitudes, and the data in Figure 7 substantiate this expectation. However, the SPSS results show that all features contribute positively in the discriminant analysis.

In summary, we have been able to improve on previous attempts to use features of the hand for access control in the areas of accuracy and user comfort. This has primarily been the result of removing the requirements on both the subject and the mechanical device that produced either a distortion of the data or discomfort for the individual. Instead, we have required the software to do all the registration and measurements. The result is a fast and reliable method that demands only a minimum of effort from the individual requesting access.

If this technique were to be packaged into a commercial product, we estimate that the throughput rate could be less than two seconds. The rate-limiting step in the process is data acquisition from the scanner through an interface and into a computer. The throughput rate of a practical interface could be in excess of 275K bytes per second. Because each image contains 256K bytes, the transfer would take about one second. This data rate is well within the state of the art of image production using a scanning linear-array system. Because the data can be analyzed a line at a time, and therefore analyzed as it is

FEATURE1	FEATURE2	FEATURE3	FEATURE4	FEATURE5	FEATURE6	FEATURE7	FEATURE8	FEATURE9
FEATURE1	1.00000							
FEATURE2	.00010	1.00000						
FEATURE3	.10101	.13153	1.00000					
FEATURE4	.22706	.13956	.72793	1.00000				
FEATURE5	.00371	-.01630	.10407	.20003	1.00000			
FEATURE6	.03333	.03972	.11396	.03939	.02033	1.00000		
FEATURE7	-.00144	.02360	.21360	.10039	.00443	.34900	1.00000	
FEATURE8	.02006	-.00344	.07300	-.03906	.06077	.27033	.30526	1.00000
FEATURE9	.11390	.00087	.00633	.11609	.13275	.30100	.37100	.42343
FEATURE10	.00511	.00333	-.03714	-.03270	-.15719	.10739	.15172	.10040
FEATURE11	.77034	.06222	.06092	.13424	.03208	-.39061	-.30514	-.15710
FEATURE12	.10040	.03055	-.00420	.00000	-.00053	-.10931	-.31052	-.20366
FEATURE13	.00193	.17950	.47752	.43000	-.01012	-.17372	-.20049	-.32994
FEATURE14	-.06370	.10701	.41902	.30742	.00346	-.21035	-.16297	-.37510
FEATURE15	-.03053	-.02049	.12101	.17016	.00262	-.14371	-.00273	-.11770
FEATURE16	-.02707	.00020	.10030	.23300	.23300	.23010	.23303	.22390
FEATURE17	.37044	.40024	.32437	.01107	.10765	.00600	.13055	.00104
FEATURE18	.10372	.12396	.30340	.40710	.14207	.40473	.30033	.40500
FEATURE19	.17330	.27775	-.01706	.11213	-.00000	-.42102	-.43130	-.47001
FEATURE20	.12866	-.02306	-.04401	-.01263	.06132	-.15900	-.03300	.11607
FEATURE21	.02040	-.10400	-.00507	-.22436	-.16901	-.00210	.00071	.15136
FEATURE22	.06436	.02134	.14130	.20795	.19330	-.02262	.14300	.00303
FEATURE23	-.06205	-.03701	-.10000	-.20000	-.22177	.33313	.10500	.17420
FEATURE24	-.01744	.07720	.05076	.07340	.07101	.40064	.10200	.20007
FEATURE25	.01009	.10942	.00132	.00900	.00057	.11301	.10201	.00091
FEATURE26	.01041	.01030	.04020	-.10742	-.13010	.12510	.05000	.10007
FEATURE27	-.02940	.00262	.06933	.02120	.00294	-.05407	.00014	.05312
FEATURE28	-.02003	-.00093	-.03933	-.10001	-.22167	.19205	.00307	.00003
FEATURE29	-.12507	.04500	.03203	.00741	.07436	.01672	-.04007	-.03743
FEATURE10	1.00000							
FEATURE11	-.00032	1.00000						
FEATURE12	-.07044	.19062	1.00000					
FEATURE13	-.10100	.10413	.24077	1.00000				
FEATURE14	-.20023	.10174	.10901	.33100	1.00000			
FEATURE15	-.06157	.07324	.02400	.13373	.20517	1.00000		
FEATURE16	.07343	-.17404	-.06350	-.09076	-.04210	.10112	1.00000	
FEATURE17	.10370	.20023	.20462	.21676	.27571	-.00097	.12015	1.00000
FEATURE18	.36323	-.16443	-.17141	-.15000	-.12170	-.21336	.46704	.30245
FEATURE19	-.24010	.30713	.07623	.30001	.42492	.13105	-.40402	.37052
FEATURE20	-.01023	.20542	.00003	-.13007	.05023	.00045	-.00450	.02047
FEATURE21	.04422	.04602	-.00257	-.10377	-.22000	-.00038	.02000	-.11100
FEATURE22	-.00462	.05260	-.00513	.02055	.17590	.19290	.00005	.10104
FEATURE23	.10575	-.25013	-.15406	-.20016	-.30707	-.22033	.10900	-.35100
FEATURE24	.11443	-.10107	-.01035	-.23948	-.10330	.03136	.17647	.31300
FEATURE25	-.01105	-.04010	.03500	-.03102	-.13400	.00209	.23577	.00159
FEATURE26	.07037	-.06705	.02000	-.13200	-.21701	-.11027	.10900	.03003
FEATURE27	-.25104	.00520	.07001	-.01004	-.04004	.02331	.10114	.11023
FEATURE28	.27070	-.14322	-.05491	-.11020	-.20555	-.14400	-.00054	-.00000
FEATURE29	-.07041	-.10201	.06304	.00977	-.01121	.30036	.03011	-.11032
FEATURE19	1.00000							
FEATURE20	.02431	1.00000						
FEATURE21	-.10474	.32000	1.00000					
FEATURE22	.00000	.24412	-.19710	1.00000				
FEATURE23	-.30050	-.23050	.02204	-.39142	1.00000			
FEATURE24	-.24001	.02100	.14350	-.00005	.15145	1.00000		
FEATURE25	-.12037	.07400	.15253	.11070	.00034	.00204	1.00000	
FEATURE26	-.15035	.04070	.33032	-.24411	.37001	.00713	.13474	1.00000
FEATURE27	-.01109	.10329	.20050	.11200	-.05400	.12400	.10000	.34720
FEATURE28	-.00000	-.23200	.03100	-.31000	.30000	.04094	.01150	.12505
FEATURE29	.04371	-.00574	-.00407	-.00201	-.01337	.00000	.00002	.04091
FEATURE20	1.00000							
FEATURE21	.17010	1.00000						

FIGURE 7 ANALYSIS OF HAND GEOMETRY FEATURES USING WITHIN-GROUPS CORRELATION MATRIX

produced, we do not believe that the data processing would be limiting. However, if line processing takes longer than scanning, multiple processors could be used with parallel multiple lines of processing working concurrently to process more than one line at a time. The need for multiple processing has not been assessed, since all work has been performed on a multi-user system with relatively slow development-type programs. However, because our software could be supported by multiple processors working in parallel, process time should not be rate-limiting.

Since the potential of this technique has been shown to be good, the next task is to refine the technique. Refinements can be made in all areas, including fine-tuning feature extraction, adding new features and removing the less significant ones, and developing methods for making authentication decisions based on these data. In addition, studies could be made of methods for optimizing the software for minimum throughput time.

VI COMPUTER PROGRAM SUMMARIES

The task of image analysis has been broken down into four discrete jobs:

- Determine the correct gray level for image thresholding using the FORTRAN program HIST.
- Threshold the image to produce a binary image using the FORTRAN program THRESH.
- Extract a list of hand-perimeter points from the binary image using the PASCAL program LINEBYLINE.
- Extract the feature values from the perimeter list using the FORTRAN program SCAN.

Each of these jobs is handled by a single independent program. A brief description of each program follows, and listings of each program appear in Appendix A. The program LINEBYLINE is quite complex, however, and Appendix B is included to describe its operation in detail.

A. HIST

This FORTRAN program computes a histogram of the frequency of occurrence of gray levels in an 8-bit gray-scale image. The program is capable of handling a 512×512 pixel image, which is the size used to represent the hand data. The image format is two pixels per 16-bit word. The record size on a DEC RK 05 disk is 256 16-bit words. The output file lists the frequency count of each gray level (0-255). A plot of the output can also be produced on a TEKTRONIX 4000-series terminal, using the PLOT 10 (TEKTRONIX) software interface installed in the Bioengineering Research Center's PDP 11/40 computer. The output of this program clearly identifies the sets of gray levels that represent either the background or the hand; this information is used to pick a discrete gray level between the two sets of levels that will best separate them. This level is used by THRESH to produce a binary image.

B. THRESH

This FORTRAN program thresholds an 8-bit gray-level image to produce a binary image. Thresholding is performed on a pixel-by-pixel basis. The input file contains two 8-bit pixels per word with a total image size of 262,144 pixels. The output file contains 16 one-bit pixels per word. Both input and output files have 256 words per record. The discrete threshold value is entered at run time.

C. LINEBYLINE

This PASCAL program produces a list of perimeter points that bound "blobs" of color opposite to that of the background in a binary image. Eight-point connectivity is checked for each point in the image. Using the large area as a criterion, the hand blob is selected from others, and only the hand perimeter is saved. This list of hand-perimeter points is output as a file of ordered points running around the perimeter of the hand. Only one pass is made through the image, and only one line is accessed at a time. The program therefore requires only slightly more dynamic memory than is required to store perimeter points. A more extensive description of this program is included as Appendix B.

D. SCAN

This FORTRAN program processes the list of perimeter points produced by LINEBYLINE and outputs numeric values that describe geometric features of the hand, as discussed earlier in this report. Included below are comments intended to help the reader interpret the program listing in Appendix A.

- The preferential landmarks on a perimeter are the fingertips. These are located by the angle algorithm on page A-20, starting with the comment lines. Landmarks are found by moving a window of size IDIS (18 points) along the perimeter and looking for an angle criterion shift of 90° in the tangent to the perimeter.
- Starting at line 174 on page A-5, calculations of finger length are made. These require the precise tip locations calculated earlier on page A-4, starting with the comment lines in the middle of the page.

- Finger widths are calculated with the code on page A-21.
- Ratios of finger lengths to widths are calculated on page A-22.
- Curvature measures are calculated with the code in the middle of page A-22, using subroutines TANAG and LINFIT that process angle-function data.
- Hand width, perimeter, and hand area are calculated at the bottom of page A-22, using the functions IDELTS and DSTNCE and the subroutine ARPRIM.

Appendix A
COMPUTER PROGRAM LISTINGS

[illegible]

33

2

1

cc

2

C

2

•

50
50

ccc

C

CCC

**C-
C**

```

200 DO 210 JREC = (STARTL-1)*2+1, (ENDL-1)*2+1, 2
D 212 IF (MOD(JREC, 20) .EQ. 1) WRITE (5, 212) JREC
      FORMAT(' JREC = ', I4)
      IF (STARTP .GT. 512) GO TO 220
      READ (LUNIN, JREC, ERR=903) (A(I), I=1, 512)
      K1 = STARTP
      K2 = MIN0(ENDP, 512)
      CALL HISTL(A, K1, K2, IH)
220 CONTINUE
      IF (ENDP .LE. 512) GO TO 210
      READ (LUNIN, JREC+1, ERR=904) (A(I), I=1, 512)
      K1 = MAX0(STARTP, 513) - 512
      K2 = ENDP - 512
      CALL HISTL(A, K1, K2, IH)
210 CONTINUE
      GO TO 500
C
C---FOR 512 POINTS/LINE (ONE LINE PER RECORD)
C
300 DO 310 JREC = STARTL, ENDL
      READ (LUNIN, JREC, ERR=903) (A(I), I=1, 512)
      CALL HISTL(A, STARTP, ENDP, IH)
310 CONTINUE
      GO TO 500
C
C---FOR 256 POINTS/LINE (2 LINES/RECORD)
C
400 DO 410 JREC = STARTL, ENDL
      READ (LUNIN, JREC, ERR=903) (A(I), I=1, 512)
C---COMPRESS DATA IN ARRAY A FROM DEANZA FORMAT
      DO 411 I = 1, 256
411 A(I) = A(2*I-1)
      CALL HISTL(A, STARTP, ENDP, IH)
410 CONTINUE
C
C---PRODUCE OUTPUT FILE IN A FORMAT ACCEPTABLE TO THE PROGRAM PLOTDATA
C
500 CONTINUE
      TIME = SECONDS(TIN)
      WRITE (5, 501) TIME
501 FORMAT(' TIME = ', F10.3, ' SECONDS')
      CLOSE (UNIT=LUNIN)
      DO 505 I = 1, 256
          G(I) = I-1
          H(I) = IH(I)
505 CONTINUE
C
      IF (ISVP .NE. 1 .AND. ISVP .NE. 3) GO TO 600
      OPEN (UNIT=LUNOUT, NAME=NAMEOUT, TYPE='NEW', ACCESS='SEQUENTIAL',
            FORM='FORMATTED', ERR=905, CARRIAGECONTROL='LIST')
C
      WRITE (4, 510) (NAMIN(I), I=1, 27), STARTL, STARTP, NL, NP
510 FORMAT(7X, 'HISTOGRAM OF ', 27A1, 4I5)
      N256 = 256
      WRITE (4, 520) N256, (G(I), H(I), I=1, 256)
520 FORMAT(14, /, 2(E15.8, 2X))
      IHAREA = 0
      DO 530 I = 1, 256
530 IHAREA = IHAREA + IH(I)
      WRITE (4, 540) IHAREA
540 FORMAT(/, ' HISTOGRAM AREA = ', I7)
      CLOSE (UNIT=LUNOUT)
C
C---PLOT THE DATA ON THE TEKTRONIX
C
600 CONTINUE
      IF (ISVP .LT. 2) GO TO 999
      ENCODE (67, 510, ITITLE, ERR=9) (NAMIN(I), I=1, 27), STARTL, STARTP, NL, NP
      CALL PLOT(G, H, NGREYL, XAXIS, YAXIS, TITLE)
C
900 STOP
C
901 WRITE (5, 911) (NAMIN(I), I=1, NCHIN)
911 FORMAT(' HIST: ERROR OPENING INPUT FILE ', 30A1)
      GO TO 999
902 WRITE (5, 912)
912 FORMAT(' HIST ERROR: # POINTS/LINE 256, 512, OR 1024 ONLY!')
      CLOSE (UNIT=LUNIN)
      GO TO 999
903 WRITE (5, 913)
913 FORMAT(' HIST ENRON: ERROR 903 READING INPUT FILE')
      CLOSE (UNIT=LUNIN)
      GO TO 999

```

```

904 WRITE (5,914)
914 FORMAT(' HIST ERROR: ERROR 904 READING INPUT FILE')
CLOSE (UNIT=LUNIN)
GO TO 999
905 WRITE (5,915) (NAMOUT(I),I=1,NCHOUT)
915 FORMAT(' HIST : ERROR CREATING OUTPUT FILE ',30A1)
GO TO 999
9 WRITE (5,916) (ITITLE(I),I=1,72)
916 FORMAT(72A1)
PAUSE 916
GO TO 999
C
END
SUBROUTINE HISTL(A,K1,K2,IN)
C
C
LOGICAL*1 A(512)
INTEGER*4 IN(256)
C
DO 100 I = K1,K2
INT = A(I)
IF (INT .LT. 0) INT = INT + 256
INDX = INT + 1
IF (IN(INDX) .EQ. 32767) GO TO 100
IN(INDX) = IN(INDX) + 1
100 CONTINUE
C
999 RETURN
END
C
C

```

PROGRAM THRESH

WRITTEN BY OREG MYERS

THIS PROGRAM THRESHOLDS AN 8-BIT GREY-LEVEL IMAGE AND PRODUCES A BINARY (1 BIT PER PIXEL) OUTPUT IMAGE. THE INPUT FILE CONTAINS 1 8-BIT PIXEL PER WORD IF THE IMAGE IS IN THE 256 X 256 DE ANZA FORMAT. OTHERWISE, THE IMAGE CONTAINS 2 PIXELS PER WORD. THE OUTPUT FILE CONTAINS 16 1-BIT PIXELS PER WORD. BOTH THE INPUT AND OUTPUT FILES HAVE 256 WORDS PER RECORD.

MODIFIED 9/24/80 TO ACCEPT COMMAND STRING INPUT
 COMMAND SYNTAX IS:
 [OUTFIL]=INFIL/LN:#:#/TH:#
 ROB HAMMON

```

LOGICAL*1 NULL,LFLG,NAMFIL(34),A(8192),B(512)
INTEGER*2 IBYTE
LOGICAL*1 BYTE
INTEGER IFLG,NL,NP,ITHR,EXSTAT
INTEGER SVTAB(9),THVAL(5),LNVAL(5)
REAL PROMPT
DATA PROMPT/'THR'/,IEXT/3REXT/,EXSTAT/1/
DATA NL/-1/,NP/-1/,ITHR/-1/
EQUIVALENCE (BYTE,IBYTE)
DATA LUNIN/3/, LUNOUT/4/, NULL/0/

```

C--GET DATA FROM TERMINAL

```

10 WRITE (5,10)
10 FORMAT('FOR INPUT FILE: TYPE #LINES, #POINTS/LINE, AND FILENAME')
10 READ (5,15) NL, NP, NCHIN, (NAMIN(I),I=1,NCHIN)
15 FORMAT(215,0,30A1)
NAMIN(NCHIN+1) = NULL

```

C Initialize GETCMD and switch descriptor tables

```

CALL ASSIGN (6,'T1',3) !Assign LUN for GETCMD
CALL INICMD (6,IEXT,PROMPT) !Set LUN and file ext,prompt
CALL CSISV (SVTAB(1),'LN',2,LNVAL) !Declare /LN switch
CALL CSISV (SVTAB(5),'TH',1,THVAL) !Declare /TH switch
CALL CSISV (LNVAL(1),'D',NL) !Declare first val for /LN:#
CALL CSISV (LNVAL(3),'D',NP) !Second value
CALL CSISV (THVAL(1),'D',ITHR) !Declare val for /TH:#

```

```

C
C Get command string and process it
10 CALL GETCMD (B,...,LFLG)
   IF (LFLG.NE. 0) GO TO 1000 !Error
   EXSTAT = 1 !Set successful
   CALL CS11(B,...,LFLG) !Compress string
   IF (LFLG.GT. 0) GO TO 1020 !Error
   IF (LFLG.EQ. 0) LFLG = '1' !Equal sign in string
   IF (LFLG.LT. 0) LFLG = '0' !No equal sign
   CALL CSIDEF(LFLG,...,'B8 ') !Set default ext
   NL = -1 !Set for check
   NP = -1
   ITHR = -1
   CALL CS12(LFLG,NAMFIL,,SWTAB) !Parse string
   IF (NAMFIL(1).EQ. 0) GO TO 1040 !Error
   IF ((NL.EQ. -1).OR. (NP.EQ. -1)) GO TO 1030 !Error
   IF (ITHR.EQ. -1) GO TO 1030 !Error
   OPEN(UNIT=LUNIN,NAME=NAMFIL,TYPE='OLD',ACCESS='DIRECT',READONLY,
1   RECORDSIZE=128,ERR=901)

C
C
   TOTPIX = FLOAT(NL) * NP
   NRECI = TOTPIX / 512
   NBLOUT = TOTPIX / 4096
   LINC = 1
   KTOT = 8
   IF (NL.NE. 256.OR. NP.NE. 256) GO TO 18
   NRECI = NRECI * 2
   LINC = 2
   KTOT = 16
18  CONTINUE

C
C Get information and open output file
   IF (LFLG.EQ. '1') GO TO 50 !Equal sign in string
   NAMFIL(25) = 'B' !Set default extension
   NAMFIL(26) = '1'
   NAMFIL(27) = 'N'
   GO TO 60
50  CALL CSIDEF('0',...NAMFIL(15),'BIN') !Set default ext
   CALL CS12('0',NAMFIL) !Parse string
   IF (NAMFIL(1).EQ. 0) GO TO 1040
C
C 20  WRITE (5,20)
C 25  FORMAT(' TYPE THRESHOLD, AND NAME FOR BINARY OUTPUT FILE')
C 25  READ (5,25) ITHR,NCHOUT,(NAMOUT(1),1=1,NCHOUT)
C 25  FORMAT(14,Q,30A1)
C 60  NAMOUT(NCHOUT+1) = NULL
C 60  OPEN(UNIT=LUNOUT,NAME=NAMFIL,TYPE='NEW',ACCESS='DIRECT',
1   RECORDSIZE=128,ERR=902,INITIALSIZE=NBLOUT)

C
C
   TIN = SECNDS(0.)
   DO 100 J = 1,NRECI,KTOT
C
C---READ IN 4096 8-BIT PIXELS
C
   DO 200 K = 1,KTOT
      KOFF = (K-1)*512
      READ (LUNIN,J+K-1) (A(KOFF+1),1=1,512)
200  CONTINUE
C
C
   LOFF = 1
   DO 300 L = 1,512
      IByte = 0
C
C-----PERFORM THE THRESHOLD IN GROUPS OF 8 PIXELS
C
   DO 400 I = 1,8
      INT = A(LOFF)
      LOFF = LOFF + LINC
      IF (INT.LT. 0) INT = INT + 256
      iByte = ishft(iByte,-1)
      if (int.ge. ithr) iByte = ior(iByte,128)
400  continue
      b(I) = iByte
   continue
300  continue
C
C---write out one record of data (4096 binary pixels)
C
   write (lunout'j/ktot+1) (b(i),i=1,512)
100  continue
C
C
   CLOSE (UNIT = LUNIN)
   CLOSE (UNIT = LUNOUT)
   TIME = SECNDS(TIN)
   WRITE (5,991) TIME
991  FORMAT(' TIME = ',F10.3)

```

```

999  GO TO 10          !Get next command
c
c
c
901  write (5,911) (NAMFIL(i),i=1,34)
911  format(' error opening input file ',30a1)
    go to 999
902  write (5,912) (NAMFIL(i),i=1,34)
912  format(' error creating output file ',30a1)
    go to 999
C
1000 IF(LFLG.EQ. '366') CALL EXST(EXSTAT) !Exit with status
1020 TYPE 1021
1021 FORMAT (' *** command syntax error ***')
    GO TO 9000
1030 TYPE 1031
1031 FORMAT (' *** error in input filespec ***')
    GO TO 9000
1040 TYPE 1041
1041 FORMAT (' *** Error in output filespec ***')
    GO TO 9000
C
9000 EXSTAT = 4          !Set for sever error
    CALL RESCMD          !Reset cmd input to top level
    GO TO 10            !Get next command
c
c
    end

```

PROGRAM LINEBYLINE(TTY); (*NSP*)

(* VERSION 46C 10/16/80

THIS VERSION PRODUCES A PERIMETER LIST OF ONE LARGE BLOB, WHICH IS SUPPOSED TO BE A HAND. FOR ROB HANMON AND THE PALMPRINT STUDY.

WRITTEN BY GREG MYERS
SRI INTERNATIONAL, J-3086
333 RAVENSWOOD AVENUE
MENLO PARK, CA. 94025

THIS ALGORITHM IS TAKEN FROM A REPORT BY GERRY AGIN AT SRI ENTITLED "IMAGE PROCESSING ALGORITHMS FOR INDUSTRIAL VISION". THIS ALGORITHM IS REFERRED TO AS "CONNECTIVITY ANALYSIS". IT SEGMENTS A BINARY IMAGE INTO "BLOBS" (CONNECTED AREAS) OF THE SAME "COLOR" (GREY LEVEL). ONLY ONE PASS IS MADE THROUGH THE IMAGE, AND ONLY ONE LINE IS ACCESSED AT A TIME (HENCE, THE PROGRAM NAME 'LINE-BY-LINE'). FEATURES OF EACH BLOB ARE COMPUTED, SUCH AS ITS AREA, CENTER OF GRAVITY, BOUNDING RECTANGLE, AND A LIST OF ITS PERIMETER POINTS.

THE LETTERS 'NSP' WITHIN A COMMENT MEAN 'NON-STANDARD PASCAL'. THE LINES MARKED 'NSP' MAY REQUIRE MODIFICATION.

THE LETTERS 'MD' WITHIN A COMMENT MEAN 'MACHINE-DEPENDENT'. THESE LINES MAY ALSO REQUIRE MODIFICATION.

BECAUSE THE COMMAND 'DISPOSE' IS NOT IN THE "SWEDISH" VERSION OF PASCAL AT SRI, 'DISPOSE' STATEMENTS ARE ENCLOSED IN COMMENT STATEMENTS AND ARE IGNORED.

READ AND WRITE STATEMENTS FROM THE USER'S TERMINAL DO NOT REQUIRE FILE SPECIFICATION IN THIS VERSION OF PASCAL. THE FILE 'TTY' IS ASSUMED.

EOLN IS AT THE BEGINNING OF AN INPUT LINE WHEN READING FROM THE TTY.

*)

```

CONST NPIXELSPERLINE = 512;      (* THIS PROGRAM HANDLES 512 X 512 IMAGES ONLY *)
    NLINES = 512;
    NBITSPERCHAR = 8;
    NCHARSPERLINE = 64;          (* = NPIXELSPERLINE / NBITSPERCHAR *)
    NCHARPERBLOCK = 512;
    BKGND = 0; INK = 1;
    (* INPUT COMMANDS *)
    ON = 'T'; OFF = 'F';
    LINEIN = 'L';
    LISTBLOBS = 'B';
    LISTPERIMS = 'P';
    LISTRUNLENGTHS = 'R';
    LISTACTIVELINESMENTS = 'A';
    DIAGNOSTICS = 'D';
    STOP = 'S';

```

```

TYPE BINARY = BKGD..INX;
CMDTYPE = SET OF 'A'..'Z';
ARRAYSINT = ARRAY[1..5] OF INTEGER;

BLOBPTR = *BLOBS;
PTRPERINSECTION = *PERINSECTION;
PERINPTR = *PERIN;

BLOBS = RECORD
    COLOR: BINARY;
    COMP: INTEGER;
    PERIN: PTRPERINSECTION;
    NPERIMPTS: INTEGER;
    AREA, XMEAN, YMEAN: REAL;
    XMIN, XMAX, YMIN, YMAX: INTEGER;
    PARENT, NEXT: BLOBPTR    END;

PERINSECTION = RECORD
    LEFT, RIGHT: PERINPTR;
    PREV, NEXT: PTRPERINSECTION    END;

PERIN = RECORD
    LINE, COL: INTEGER;
    NEXT: PERINPTR    END;

SEGPtr = *SEGMENT;
SEGMENT = RECORD
    STARTCOL, ENDCOL: INTEGER;
    BLOB: BLOBPTR;
    NEXT: SEGPtr    END;

PTRRUNLENGTH = *RUNLENGTHS;
RUNLENGTHS = RECORD
    STARTCOL, ENDCOL: INTEGER;
    NEXT: PTRRUNLENGTH    END;

(* GLOBAL VARIABLES *)
VAR BLOB, BLOBSDONE: BLOBPTR;
    RECYCLEDPtr: PERINPTR;
    ARRAYP: ARRAY[1..11] OF PERINPTR;
    ACTIVELINE, PREVSEG, CURRSEG, SEG: SEGPtr;
    NEWLINE, NEWSEG, LASTNEWSEG: PTRRUNLENGTH;
    CMD, ONOFF: CHAR;
    CMDSET: CMDTYPE;
    INTERACTIVE: BOOLEAN;
    LINENUM, COL, NEWCOMPNUM, NLREAD, NTIMES: INTEGER;
    TRACEBLOBS, TRACEPERIMS, TRACEACTIVELINESEGMENTS, TRACERUNLENGTHS: CHAR;
    TRACEDIAGNOSTICS: CHAR;
    INPUTIMAGE, DATAFILE, PERINFILE: TEXT;
    INPUTNAME, DATANAME, PERINNAME: ARRAY[1..30] OF CHAR;    (*MD*)
    BIT: ARRAY[1..NPIXELSPERLINE] OF BINARY;
    I, CHARTOT, LINEGROUP: INTEGER;
    TYPELINENUMS: BOOLEAN;
    HANDONLY, BLOBWRITTEN: BOOLEAN;    AREATH: REAL;    IAREATH: INTEGER;
    TRACEPARENT, RECURS: BOOLEAN;
    NSKIP: INTEGER;

(*OP* NEW PAGE *)
PROCEDURE PAUSE;
VAR DUMMYCHAR: CHAR;
BEGIN
    WRITELN('TYPE ANY CHARACTER TO CONTINUE');
    READLN;
    READ(DUMMYCHAR)
END;    (* PAUSE *)

PROCEDURE TYPERUNLENGTHS;
VAR RLPTR: PTRRUNLENGTH;
BEGIN
END;    (* TYPERUNLENGTHS *)

PROCEDURE TYPEACTIVELINESEGMENTS;
VAR ALSPTR: SEGPtr;
BEGIN
END;    (* TYPEACTIVELINESEGMENTS *)

```

```
(*SP* NEW PAGE *)
PROCEDURE WRITEBLOB(VAR DEVICE: TEXT; VAR BLOB: BLOBPTR);
BEGIN
END; (* WRITEBLOB *)
```

```
PROCEDURE TYPEBLOBS(VAR DEVICE: TEXT);
VAR ALSPTR: SEGPTR;
BEGIN
END; (* TYPEBLOBS *)
```

```
(*SP* NEW PAGE *)
PROCEDURE DIRPOINT(VAR DIR, X1, Y1, X2, Y2: INTEGER);
```

```
  (* DIRECTIONS:
      X
      -1 0 1
  Y  -1 3 2 1
      0 4 0
      1 5 6 7 *)
  BEGIN
    CASE DIR OF
      3,2,1: Y2 := Y1-1;
      4,0 : Y2 := Y1;
      5,6,7: Y2 := Y1+1 END;

      3,4,5: X2 := X1-1;
      2,6 : X2 := X1;
      0,1,7: X2 := X1+1 END;
  END; (* DIRPOINT *)
```

```
PROCEDURE UNPACKS(VAR XX: INTEGER; VAR DIR: ARRAYSINT);
```

```
VAR I, X, XDIV4096, XDIV512, XDIV64, XDIV8: INTEGER;
BEGIN
```

```
  X := -XX;
  XDIV4096 := X DIV 4096;
  XDIV512 := X DIV 512;
  XDIV64 := X DIV 64;
  XDIV8 := X DIV 8;
  DIR[1] := X - XDIV8*8;
  DIR[2] := XDIV8 - XDIV64*8;
  DIR[3] := XDIV64 - XDIV512*8;
  DIR[4] := XDIV512 - XDIV4096*8;
  DIR[5] := XDIV4096;
END; (* UNPACKS *)
```

```
(*SP* NEW PAGE *)
PROCEDURE UNPACKPERIMS(VAR PTR, UPPTR, UPPTL: PERIMPTR);
```

```
VAR DIR: ARRAYSINT;
    I, J: INTEGER;
```

```
  BEGIN
    ARRAYP[1].LINE := PTR.LINE;
    ARRAYP[1].COL := PTR.COL - NPIXELSPERLINE;

    UNPACKS(PTR.NEXT.LINE, DIR);

    FOR I := 1 TO 5 DO BEGIN
      J := I + 1;
      DIRPOINT(DIR[I], ARRAYP[1].COL, ARRAYP[1].LINE, ARRAYP[J].COL, ARRAYP[J].LINE);
      END; (* FOR *)

    UNPACKS(PTR.NEXT.COL, DIR);
    IF TRACEDIAGNOSTICS = ON THEN FOR I := 1 TO 5 DO WRITELN(DIR[I]; 4);

    FOR I := 6 TO 10 DO BEGIN
      J := I + 1;
      DIRPOINT(DIR[I-5], ARRAYP[1].COL, ARRAYP[1].LINE, ARRAYP[J].COL, ARRAYP[J].LINE);
      END; (* FOR *)

    UPPTR := ARRAYP[1];
    UPPTL := ARRAYP[1];
  END; (* UNPACKPERIMS *)
```

```
(*SP* NEW PAGE *)
FUNCTION CONNECTED(VAR POINT1, POINT2: PERIMPTR): BOOLEAN;
```

```

BEGIN
  IF (ABS(POINT1.LINE-POINT2.LINE) <= 1) AND
     (ABS(POINT1.COL-POINT2.COL) <= 1) THEN CONNECTED := TRUE
  ELSE CONNECTED := FALSE;
END;

PROCEDURE WRITEPOINTS(VAR DEVICE: TEXT;
                     VAR RIGHT,LEFT: PERIMPTR;
                     VAR NPERIMPTS: INTEGER);

VAR PTR,PTR2,UPPTRR,UPPTRL: PERIMPTR;
BEGIN
  (* IF (TRACEDIAGNOSTICS = ON) AND RECURS THEN BEGIN
    Writeln('RECURS=TRUE');
    PTR2 := RIGHT;
    WHILE PTR2 <> LEFT DO BEGIN
      Writeln(PTR2.LINE:4,PTR2.COL:4);
      PTR2 := PTR2.NEXT; END;
    Writeln(PTR2.LINE:4,PTR2.COL:4); END; *)
  PTR := RIGHT;
  NPERIMPTS := NPERIMPTS + 1;
  WITH PTR DO Writeln(DEVICE,LINE:4,' ',COL:4);
  WHILE PTR <> LEFT DO BEGIN
    IF PTR.NEXT.COL > NPIXELSPERLINE THEN BEGIN
      IF RECURS THEN BEGIN Writeln('2ND RECURSION'); PAUSE END;
      UNPACKPERIMS(PTR.NEXT,UPPTRR,UPPTRL);
      (* IF TRACEDIAGNOSTICS = ON THEN Writeln('WRITEPOINTS -- A');
        IF TRACEDIAGNOSTICS = ON THEN
          RECURS := TRUE; *)
      (* PTR2 := UPPTRR;
        WHILE PTR2 <> UPPTRL DO BEGIN
          Writeln(DEVICE,PTR2.LINE:4,PTR2.COL:4);
          PTR2 := PTR2.NEXT; END;
        Writeln(DEVICE,PTR2.LINE:4,PTR2.COL:4); *)
        WRITEPOINTS(DEVICE,UPPTRR,UPPTRL,NPERIMPTS);
        (* IF TRACEDIAGNOSTICS = ON THEN Writeln('WRITEPOINTS -- B');
          IF TRACEDIAGNOSTICS = ON THEN RECURS := FALSE; *)
        PTR := PTR.NEXT.NEXT; END
      ELSE BEGIN
        (* IF NOT CONNECTED(PTR,PTR.NEXT) THEN BEGIN
          Writeln('POINTS NOT CONNECTED');
          Writeln(DEVICE,'POINTS NOT CONNECTED'); PAUSE; END; *)
        (* SKIP UNNECESSARY PERIMETER POINTS *)
        IF (PTR.NEXT <> LEFT) AND CONNECTED(PTR,PTR.NEXT.NEXT)
          THEN PTR := PTR.NEXT;
        PTR := PTR.NEXT;
        NPERIMPTS := NPERIMPTS + 1;
        (* IF TRACEDIAGNOSTICS = ON THEN
          WITH PTR DO Writeln(LINE:4,' ',COL:4); *)
        WITH PTR DO Writeln(DEVICE,LINE:4,' ',COL:4);
        END; (* ELSE *)
      END; (* WHILE *)
    END; (* WRITEPOINTS *)

  (***) NEW PAGE **)
PROCEDURE WRITEPERIMS(VAR DEVICE: TEXT; VAR BLOB: BLOBPTR);
VAR PSPTR: PTRPERIMSECTION;
BEGIN
  WITH BLOB DO
    IF PERIM <> NIL THEN BEGIN
      NPERIMPTS := 0;
      IF NOT HANDONLY THEN Writeln(DEVICE,' ');
      IF NOT HANDONLY THEN Writeln(DEVICE,'BLOB COMPONENT # ',COMP:4);
      PSPTR := PERIM;
      REPEAT
        IF NOT HANDONLY THEN Writeln(DEVICE,' ');
        IF NOT HANDONLY THEN Writeln(DEVICE,'LINE COL');
        WITH PSPTR DO WRITEPOINTS(DEVICE,RIGHT,LEFT,NPERIMPTS);
        PSPTR := PSPTR.NEXT
      UNTIL PSPTR = PERIM;
      END (* IF *)
    END; (* WRITEPERIMS *)

PROCEDURE TYPEPERIMS(VAR DEVICE:TEXT);
VAR ALSPTR: SEGPTR;
BEGIN
  Writeln(DEVICE,' ');
  Writeln(DEVICE,'BLOB PERIMETERS');
  ALSPTR := ACTIVELINE;
  WHILE ALSPTR <> NIL DO BEGIN
    WRITEPERIMS(DEVICE,ALSPTR.BLOB);
    ALSPTR := ALSPTR.NEXT; END;
  END; (* TYPEPERIMS *)

```

```

(*SP* NEW PAGE *)
PROCEDURE NEW(VAR NEWPTR: PERIMPTR);
(*NSP*)
(* THIS ROUTINE IS NEEDED ONLY IF THE COMMAND 'DISPOSE' IS NOT AVAILABLE
IN THIS VERSION OF PASCAL *)
BEGIN
  IF RECYCLEDPTR <> NIL THEN BEGIN
    NEWPTR := RECYCLEDPTR;
    RECYCLEDPTR := RECYCLEDPTR.NEXT;
    NEWPTR.NEXT := NIL;
  END;
  ELSE NEW(NEWPTR);
END; (* NEW *)

PROCEDURE DELETEPERIMS(VAR RIGHT, LEFT: PERIMPTR);
VAR POINT: PERIMPTR;
BEGIN
  (* ADD THE LIST OF PERIMETER POINTS OF THE BLOB TO THE LIST OF
  RECYCLED PERIMETER POINTS. THIS SECTION OF CODE SHOULD
  BE USED IF THE COMMAND 'DISPOSE' IS NOT IMPLEMENTED IN THIS VERSION
  OF PASCAL *)
  LEFT.NEXT := RECYCLEDPTR;
  RECYCLEDPTR := RIGHT;

  (* DISPOSE OF PERIMETER POINTS AND PERIMETER SECTION. THIS SECTION OF CODE
  CAN BE USED ONLY IF THE COMMAND 'DISPOSE' IS IMPLEMENTED. *)
  (* WHILE RIGHT <> LEFT DO BEGIN
    POINT := RIGHT;
    RIGHT := RIGHT.NEXT;
    DISPOSE(POINT) END; *)
  DISPOSE(RIGHT)
END; (* DELETEPERIMS *)

(*SP* NEW PAGE *)
FUNCTION DIRECTION(VAR X1,Y1,X2,Y2: INTEGER): INTEGER;

  (* DIRECTIONS:
      X
      -1 0 1
  Y  -1 3 2 1
      0 4 0
      1 5 6 7 *)

  VAR DELTAY: INTEGER;
  BEGIN
    DELTAY := Y2 - Y1;
    CASE X2 - X1 OF
      -1: CASE DELTAY OF
          -1: DIRECTION := 3;
           0: DIRECTION := 4;
           1: DIRECTION := 5 END;
      0: CASE DELTAY OF
          -1: DIRECTION := 2;
           1: DIRECTION := 6 END;
      1: CASE DELTAY OF
          -1: DIRECTION := 1;
           0: DIRECTION := 0;
           1: DIRECTION := 7 END;
    END; (* CASE DELTAY *)
  END; (* DIRECTION *)

(*SP* NEW PAGE *)
PROCEDURE COMPACTPOINTS(VAR PTR, PTRB: PERIMPTR; VAR I: INTEGER);
VAR PREVLINE, PREVCOL, POSITION, DIR: INTEGER;
BEGIN
  WITH PTR DO BEGIN
    IF I = 0 THEN BEGIN
      PTRB.LINE := 0;
      PTRB.COL := 0;
      PTRB.NEXT := NIL;
    END
    ELSE IF I < 6 THEN BEGIN
      IF I = 1 THEN POSITION := 1 ELSE POSITION := POSITION * 6;
      PTRB.LINE := PTRB.LINE - POSITION * DIRECTION(PREVCOL, PREVLINE, COL, LINE);
    END
  END

```

```

ELSE BEGIN
  IF I = 6 THEN POSITION := 1 ELSE POSITION := POSITION * 8;
  PTRB.COL := PTRB.COL - POSITION * DIRECTION(PREVCOL,PREVLINE,COL,LINE);
END;
PREVLINE := LINE;
PREVCOL := COL;
END; (* WITH *)
END; (* COMPACTPOINTS *)

(*P* NEW PAGE *)
PROCEDURE PKPERIMS(VAR PERIM: PTRPERINSECTION);
VAR PTR,PTRA,PTRB: PTRPERINSECTION;
FIRSTGROUPOF10: BOOLEAN; (* NOT USED RIGHT NOW *)
NPERIMPTS, I: INTEGER;
BEGIN
  PTR := PERIM.RIGHT;
  I := 0;
  FIRSTGROUPOF10 := TRUE;
  IF PTR <> PERIM.LEFT THEN
    REPEAT BEGIN
      IF (PTR.LINE < LINENUM-1) AND (PTR.COL > 0)
        AND (PTR.COL <= NPIXELSPERLINE) THEN BEGIN
        IF I = 0 THEN BEGIN
          PTRA := PTR;
          NEW(PTRB) END;
          COMPACTPOINTS(PTR,PTRB,I);
          I := I + 1;
          IF I = 11 THEN BEGIN
            (* TO IDENTIFY THAT COMPACTED RECORDS FOLLOW *)
            PTRA.COL := PTRA.COL + NPIXELSPERLINE;
            (* INSERT COMPACTED PERIMETER RECORD INTO THE LIST *)
            PTRB.NEXT := PTR.NEXT;
            DELETERPERIMS(PTRA.NEXT,PTR);
            PTRA.NEXT := PTRB;
            PTR := PTRB;
            I := 0;
            END; (* IF I = 11 *)
            END (* IF 3 CONDITIONS THEN *)
            ELSE IF I > 0 THEN BEGIN
              (* AN INCOMPLETE COMPACTION OCCURED -- RESET VARIABLES *)
              I := 0;
              FIRSTGROUPOF10 := TRUE;
              DELETERPERIMS(PTRB,PTRB) END;
              PTR := PTR.NEXT;
              END (* REPEAT BEGIN *)
            UNTIL PTR = PERIM.LEFT;
          END; (* PKPERIMS *)

PROCEDURE PACKPERIMS;
VAR NPERIMPTS: INTEGER;
BEGIN
  SEQ := ACTIVELINE;
  WHILE SEQ <> NIL DO BEGIN
    WITH SEQ.BLOB DO
      IF PERIM <> NIL THEN BEGIN
        PKPERIMS(PERIM);
        PERIM := PERIM.NEXT END;
      SEQ := SEQ.NEXT END;
    END; (* PACKPERIMS *)

(*P* NEW PAGE *)
FUNCTION ENDOFBLOCK: BOOLEAN;
BEGIN
  IF CHARTOT = NCHARPERBLOCK THEN BEGIN
    CHARTOT := 0;
    ENDOFBLOCK := TRUE END
  ELSE ENDOFBLOCK := FALSE;
  CHARTOT := CHARTOT + 1;
END; (* ENDOFBLOCK *)

PROCEDURE GETLINE;
(* GET A LINE OF BINARY IMAGE DATA FROM AN INPUT FILE. *)
VAR INPUTBYTE,BOBL: CHAR;
I,J,INDX,BITS: INTEGER;
BEGIN

```

```

(* READ IN ONE LINE OF DATA FROM THE INPUT FILE *)
(* UNPACK THE BINARY DATA: IT IS ASSUMED THAT THE DATA IS INTERPRETED AS
   A SERIES OF 8-BIT CHARACTERS *)

INDX := 0;
FOR I := 1 TO NCHARSPERLINE DO BEGIN
  IF ENDOFBLOCK THEN READ(INPUTIMAGE,EOBL);
  READ(INPUTIMAGE,INPUTBYTE);
  BITS := ORD(INPUTBYTE);
  FOR J := 1 TO NBITSPECHAR DO BEGIN
    INDX := INDX + 1;
    IF ODD(BITS) THEN BIT(INDX) := INK;
    ELSE BIT(INDX) := BKGND;
  END;
  BITS := BITS DIV 2;
END;
END; (* GETLINE *)
PROCEDURE SETLINE;
  (* SET UP STARTING RUNLENGTH RECORD *)
BEGIN
  IF LINENUM = 1 THEN BEGIN
    NEW(NEWLINE);
    NEWLINE.NEXT := NIL; END;
    NEWLINE.STARTCOL := MININT+1;
    NEWLINE.ENDCOL := MAXINT-1;
    NEWSEG := NEWLINE;
    LASTNEWSEG := NEWLINE;
  END; (* SETLINE *)

PROCEDURE ADDRUNLENGTH(COLUMN: INTEGER);
VAR NEWESTSEG: PTRRUNLENGTH;
BEGIN
  IF NEWSEG.NEXT <> NIL
    THEN NEWESTSEG := NEWSEG.NEXT (* ADVANCE NEW SEGMENT POINTER IF
      A SEGMENT ALREADY EXISTS *)
    ELSE BEGIN
      NEW(NEWESTSEG); (* CREATE A NEW SEGMENT IF IT DOESN'T EXIST *)
      NEWSEG.NEXT := NEWESTSEG; (* CONNECT NEWEST SEGMENT TO LIST *)
      NEWESTSEG.NEXT := NIL; END;
      NEWSEG.ENDCOL := COLUMN - 1;
      NEWESTSEG.STARTCOL := COLUMN;
      NEWESTSEG.ENDCOL := MAXINT-1;
      NEWSEG := NEWESTSEG (* UPDATE SEGMENT POINTER TO THE NEWEST SEGMENT *)
    END; (* ADDRUNLENGTH *)

PROCEDURE CREATERUNLENGTHS;
(* FOR EACH TRANSITION, CREATE A NEW RECORD FOR THE RUNLENGTH SEGMENT *)
VAR I: INTEGER;
BEGIN
  IF BIT(1) = INK THEN ADDRUNLENGTH(1);
  FOR I := 2 TO NPIXELSPERLINE DO
    IF BIT(I-1) <> BIT(I) THEN ADDRUNLENGTH(I);
  IF BIT(NPIXELSPERLINE) = INK THEN ADDRUNLENGTH(NPIXELSPERLINE+1);
  LASTNEWSEG := NEWSEG;
  IF TRACERUNLENGTHS = ON THEN TYPERUNLENGTHS;
END; (* CREATERUNLENGTHS *)

(* ** NEW PAGE *)
PROCEDURE ADDEIGHTPERINPOINT(VAR PERIN:PTRPERINSECTION; LINE,COL:INTEGER);
(* THIS PROCEDURE ADDS ONE POINT TO THE PERIMETER LIST. THE NEW POINT IS
   INSERTED BEFORE THE POINT DESIGNATED BY PERIN.RIGHT *)
VAR NEWPOINTER: PERIMPTR;
BEGIN
  NEW(NEWPOINTER);
  NEWPOINTER.LINE := LINE;
  NEWPOINTER.COL := COL;
  NEWPOINTER.NEXT := PERIN.RIGHT;
  PERIN.RIGHT := NEWPOINTER;
END; (* ADDEIGHTPERINPOINT *)

PROCEDURE ADDEFTPERINPOINT(VAR PERIN:PTRPERINSECTION; LINE,COL:INTEGER);
(* THIS PROCEDURE ADDS ONE POINT TO THE PERIMETER LIST. THE NEW POINT IS
   INSERTED AFTER THE POINT DESIGNATED BY PERIN.LEFT *)

```

```

VAR NEWPTR: PERIMPTR;
BEGIN
  NEW(NEWPTR);
  NEWPTR.LINE := LINE;
  NEWPTR.COL := COL;
  NEWPTR.NEXT := NIL;
  PERIM.LEFT.NEXT := NEWPTR; (* CONNECT PREVIOUS POINT AND NEW POINT *)
  PERIM.LEFT := NEWPTR; (* RESET LEFT POINTER *)
END; (* ADDELEFTPERIMPOINT *)

(*SP. NEW PAGE *)
PROCEDURE INSERTSEGMENT(VAR STARTCOL,ENDCOL:INTEGER);

VAR SURRBLOB: BLOBPTR;
SURRCOMP: INTEGER;
SURRCOLOR: BINARY;
A,B: SEGPTR;
COL: INTEGER;
NEWPERIM: PTRPERIMSECTION;

BEGIN
  IF TRACEDIAGNOSTICS = ON THEN WRITELN('BEGIN INSERTION ');
  SURRBLOB := PREVSEG.BLOB; (* SURROUNDING BLOB *)
  SURRCOMP := SURRBLOB.COMP;
  SURRCOLOR := SURRBLOB.COLOR;

  (* CREATE A NEW BLOB FOR THE NEW SEGMENT *)
  NEW(BLOB);
  WITH BLOB DO BEGIN
    IF SURRCOLOR = BKGD THEN COLOR := INK
    ELSE COLOR := BKGD;
    NEWCOMPNUM := NEWCOMPNUM + 1;
    COMP := NEWCOMPNUM;
    AREA := 0.0;
    XMEAN := 0.0;
    YMEAN := 0.0;
    XMIN := STARTCOL;
    XMAX := ENDCOL;
    YMIN := LINENUM;
    YMAX := LINENUM;
    PARENT := SURRBLOB;
    NEXT := NIL;
    IF COLOR = BKGD THEN PERIM := NIL (* NO PERIMETER LIST FOR BACKGROUND BLOB *)
    ELSE BEGIN
      NEW(PERIM);
      PERIM.PREV := PERIM;
      PERIM.NEXT := PERIM;
      PERIM.RIGHT := NIL;
      ADDELEFTPERIMPOINT(PERIM,LINENUM,STARTCOL); (* ADD FIRST POINT *)
      PERIM.LEFT := PERIM.RIGHT;
      FOR COL := STARTCOL+1 TO ENDCOL DO
        ADDELEFTPERIMPOINT(PERIM,LINENUM,COL) (* ADD REMAINING POINTS *)
      END (* ELSE *)
    END; (* WITH BLOB *)
  IF TRACEDIAGNOSTICS = ON THEN WRITELN('DEBUG INSERTSEGMENT 2');

  (* FORM A NEW PERIMETER SECTION FOR THE SURROUNDING BLOB *)
  WITH SURRBLOB DO
    IF COLOR <> BKGD THEN BEGIN
      (* CREATE A NEW PERIMETER SECTION AND INSERT IT BEFORE THE
      CURRENT PERIMETER SECTION *)
      NEW(NEWPERIM);
      NEWPERIM.NEXT := PERIM;
      NEWPERIM.PREV := PERIM.PREV;
      PERIM.PREV.NEXT := NEWPERIM;
      PERIM.PREV := NEWPERIM;

      (* MOVE PART OF THE RIGHT END OF THE CURRENT PERIMETER SECTION TO THE
      NEW PERIMETER SECTION *)
      WITH PERIM DO BEGIN
        NEWPERIM.RIGHT := RIGHT; (* SET RIGHT POINTER OF NEW SECTION *)
        IF TRACEDIAGNOSTICS = ON THEN WRITELN('DEBUG INSERTSEGMENT 2A');
        WHILE (RIGHT.LINE <> LINENUM-1) OR (RIGHT.COL <> ENDCOL)
          DO BEGIN IF TRACEDIAGNOSTICS = ON THEN WRITELN(RIGHT.LINE,RIGHT.COL,LINENUM-1,ENDCOL);
            RIGHT := RIGHT.NEXT; END;
        IF TRACEDIAGNOSTICS = ON THEN WRITELN('DEBUG INSERTSEGMENT 2B');
        NEWPERIM.LEFT := RIGHT; (* SET LEFT POINTER OF NEW SECTION *)
        IF (RIGHT.NEXT.LINE = LINENUM-1) AND (RIGHT.NEXT.COL > RIGHT.COL) THEN RIGHT := RIGHT.NEXT;
        WHILE (RIGHT.NEXT.LINE = LINENUM-1) AND (RIGHT.NEXT.COL > RIGHT.COL) DO BEGIN
          IF TRACEDIAGNOSTICS = ON THEN WRITELN(RIGHT.LINE,RIGHT.COL,LINENUM-1);
          RIGHT := RIGHT.NEXT;
          (* SET THE RIGHT POINTER OF THE CURRENT
          SECTION TO THE ENDPOINT OF THE LINE
          SEGMENT ON THE PREVIOUS LINE THAT IS
          PART OF THE SURROUNDING BLOB *)
        END;
      END;
    END;
  END;

```

```

        IF RIGHT = NEWPERIM.LEFT THEN BEGIN
            COL := RIGHT.COL;
            RIGHT := RIGHT.NEXT;
            ADDRIGHTPERINPOINT(PERIM,LINENUM-1,COL); END;
        NEWPERIM.LEFT.NEXT := NIL;
        END; (* WITH PERIM *)
        PERIM := NEWPERIM
    END; (* WITH SURRBLOB *)

(* INSERT 2 SEGMENTS IN THE ACTIVE LINE IMMEDIATELY BEFORE THE CURRENT
SEGMENT POINTER *)

    IF TRACEDIAGNOSTICS = ON THEN WRITELN('DEBUG INSERTSEGMENT 3');
    NEW(A); NEW(B);
    A.STARTCOL := STARTCOL;
    B.STARTCOL := ENDCOL+1;
    A.ENDCOL := ENDCOL;
    B.ENDCOL := CURRSEG.STARTCOL - 1;
    A.BLOB := BLOB;
    B.BLOB := SURRBLOB;
    PREVSEG.NEXT := A;
    A.NEXT := B;
    B.NEXT := CURRSEG;
    CURRSEG := A; (* RESET CURRENT SEGMENT POINTER *)

    IF TRACEACTIVELINESEGMENTS = ON THEN TYPEACTIVELINESEGMENTS;
    IF TRACEPERIMS = ON THEN BEGIN
        WRITEPERIMS(TTY,SURRBLOB); WRITEPERIMS(TTY,BLOB) END;
    END; (* INSERTSEGMENT *)

(*P* NEW PAGE *)
PROCEDURE ADDBLOB(VAR BLOB,TERMBLOB: BLOBPTR);

    (* THIS IS A RECURSIVE PROCEDURE FOR ADDING A BLOB TO THE END OF
    THE LIST OF COMPLETED BLOBS *)

    BEGIN
        IF BLOB = NIL THEN BLOB := TERMBLOB
        ELSE ADDBLOB(BLOB.NEXT,TERMBLOB)
    END; (* ADDBLOB *)

PROCEDURE RECORDBLOB(VAR TERMBLOB: BLOBPTR);

    BEGIN
        IF (HANDONLY AND (TERMBLOB.AREA < AREATH)) OR BLOBWRITTEN THEN ELSE
            WRITEPERIMS(PERINFILE,TERMBLOB);
            IF HANDONLY AND (NOT BLOBWRITTEN) AND (TERMBLOB.AREA >= AREATH) THEN BEGIN
                BLOBWRITTEN := TRUE;
                WRITELN('BLOB OF HAND IS DONE'); END;
            WITH TERMBLOB.PERIM DO DELETERIMS(RIGHT,LEFT);
            (* DISPOSE(TERMBLOB.PERIM); *) (*NSP*)
            WRITEBLOB(DATAFILE,TERMBLOB);
            ADDBLOB(BLOSDONE,TERMBLOB);
        END; (* RECORDBLOB *)

(*P* NEW PAGE *)
PROCEDURE DELETESGMENT;

    VAR TERMBLOB,RIGHTBLOB,LEFTBLOB,REPLACBLOB,ABSORBBLOB: BLOBPTR;
        TERMCOMP,RIGHTCOMP,LEFTCOMP,REPLACCOMP,ABSORBCOMP: INTEGER;
        RIGHTCOLOR: BINARY;
        NEWAREA: REAL;
        OLDPERIM: PTRPERINSECTION;
        PERIMLPOINT,PERIMRPOINT,INTERIORPOINT,NEXTPOINT: PERIMPTR;
        COL,ABSORBRCOL: INTEGER;

    BEGIN
        IF TRACEDIAGNOSTICS = ON THEN WRITELN('BEGIN DELETING');
        TERMBLOB := CURRSEG.BLOB; (* TERMINATING BLOB *)
        TERMCOMP := TERMBLOB.COMP;
        RIGHTBLOB := CURRSEG.NEXT.BLOB;
        RIGHTCOMP := RIGHTBLOB.COMP;
        RIGHTCOLOR := RIGHTBLOB.COLOR;
        LEFTBLOB := PREVSEG.BLOB;
        LEFTCOMP := LEFTBLOB.COMP;
        IF RIGHTCOMP = 0
            THEN BEGIN
                REPLACBLOB := RIGHTBLOB; ABSORBBLOB := LEFTBLOB;
                REPLACCOMP := RIGHTCOMP; ABSORBCOMP := LEFTCOMP END
            ELSE BEGIN
                REPLACBLOB := LEFTBLOB; ABSORBBLOB := RIGHTBLOB;
                REPLACCOMP := LEFTCOMP; ABSORBCOMP := RIGHTCOMP END;
        IF TRACEDIAGNOSTICS = ON THEN BEGIN
            WRITELN('REPLACING COMPONENT = ',REPLACCOMP,
                'ABSORBED COMPONENT = ',ABSORBCOMP);
            WRITELN('RIGHTCOLOR = ',RIGHTCOLOR,' RIGHT COMP = ',RIGHTCOMP) END;

```

```

(* CONNECT THE ENDS OF TWO PERIMETER SECTIONS OF THE TERMINATING BLOB IF ITS
COLOR IS NOT BACKGROUND *)
WITH TERMBLOB: DO
  IF COLOR <> BKGND THEN BEGIN
    (* ADD INTERIOR POINTS ON THE BOTTOM LINE OF THE TERMINATING BLOB TO THE
    LIST OF PERIMETER POINTS *)
    FOR COL := CURRSEG.STARTCOL+1 TO CURRSEG.ENDCOL-1 DO
      ADDLEFTPERIMPOINT(PERIM,LINENUM-1,COL);
    (* IF THERE IS MORE THAN ONE PERIMETER POINT IN THE BLOB AND
    IF THERE IS ONLY ONE POINT ON THE BOTTOM LINE OF THE TERMINATING BLOB,
    IT IS AT BOTH THE LEFT END OF THE CURRENT PERIMETER SECTION AND THE
    RIGHT END OF THE NEXT PERIMETER SECTION; DELETE THE POINT AT THE
    RIGHT END OF THE NEXT PERIMETER SECTION *)
    IF (PERIM.LEFT <> PERIM.RIGHT) AND (CURRSEG.STARTCOL = CURRSEG.ENDCOL) THEN
      WITH PERIM.NEXT: DO RIGHT := RIGHT.NEXT;
    IF TRACEDIAGNOSTICS = ON THEN WRITELN('DISPOSE CHECK 1');
    IF TRACEPERIMS = ON THEN WRITEPERIMS(TTY,TERMBLOB);
    (* CONNECT THE LEFT END OF THE CURRENT PERIMETER SECTION WITH THE RIGHT
    END OF THE NEXT PERIMETER SECTION *)
    WITH PERIM: DO BEGIN
      LEFT.NEXT := NEXT.RIGHT;
      NEXT.RIGHT := RIGHT; (* RESET THE RIGHT END POINTER OF THE
      NEXT PERIMETER SECTION *)
      PREV.NEXT := NEXT; (* CONNECT THE PREVIOUS SECTION *)
      NEXT.PREV := PREV END; (* WITH THE NEXT SECTION *)
    OLDPERIM := PERIM;
    PERIM := PERIM.NEXT; (* ADVANCE THE PERIMETER SECTION POINTER *)
    (* IF THERE IS MORE THAN ONE PERIMETER SECTION, THEN DISPOSE OF THE OLD ONE;
    OTHERWISE, ADD THE BLOB TO THE LIST OF BLOBS DONE *)
    IF PERIM <> OLDPERIM
      THEN (* DISPOSE(OLDPERIM) *)
      ELSE RECORDBLOB(TERMBLOB);
    END; (* IF, WITH *)
    IF TRACEDIAGNOSTICS = ON THEN WRITELN('DISPOSE CHECK 2');
    IF TRACEPERIMS = ON THEN WRITEPERIMS(TTY,TERMBLOB);
  )
(* CONNECT THE ENDS OF THE TWO PERIMETER SECTIONS OF THE REPLACING BLOB IF ITS
COLOR IS NOT BACKGROUND *)
WITH REPLACBLOB: DO BEGIN
  IF COLOR <> BKGND THEN BEGIN
    (* CONNECT LEFT END POINT OF ABSORBED SECTION TO A PERIMETER POINT IN THE
    REPLACING SECTION *)
    PERIMLPOINT := PERIM.RIGHT;
    WHILE PERIMLPOINT.COL <> ABSORBLOB.PERIM.LEFT.COL-1 DO
      PERIMLPOINT := PERIMLPOINT.NEXT;
    ABSORBLOB.PERIM.LEFT.NEXT := PERIMLPOINT;
    IF TRACEDIAGNOSTICS = ON THEN
      WRITELN('PERIMLPOINT = ',PERIMLPOINT.LINE:4,', ',PERIMLPOINT.COL:4);
    IF TRACEPERIMS = ON THEN WRITEPERIMS(TTY,REPLACBLOB);
    PERIMRPOINT := PERIM.RIGHT;
    ABSORBCOL := ABSORBLOB.PERIM.NEXT.RIGHT.COL;
    IF PERIM.RIGHT.COL > ABSORBCOL+1 THEN
      WHILE PERIMRPOINT.COL <> ABSORBCOL+1 DO PERIMRPOINT := PERIMRPOINT.NEXT
    ELSE IF PERIM.RIGHT.COL < ABSORBCOL-1 THEN
      FOR COL := ABSORBCOL-1 DOWNTO PERIM.RIGHT.COL+1 DO
        ADDRIGHTPERIMPOINT(ABSORBLOB.PERIM.NEXT,LINENUM-1,COL);
    IF TRACEDIAGNOSTICS = ON THEN WRITELN('DISPOSE DBUG 2B');
    IF TRACEPERIMS = ON THEN WRITEPERIMS(TTY,REPLACBLOB);
    (* DELETE POINTS IN THE MIDDLE OF THE LINE SEGMENT CONNECTING THE TWO
    BLOBS WHICH ARE NOT NOW PERIMETER POINTS *)
    IF TRACEDIAGNOSTICS = ON THEN
      WRITELN('PERIMRPOINT = ',PERIMRPOINT.LINE:4,', ',PERIMRPOINT.COL:4);
    IF (PERIMLPOINT <> PERIMRPOINT) AND (PERIMLPOINT <> PERIMRPOINT.NEXT) THEN BEGIN
      NEXTPOINT := PERIMRPOINT.NEXT;
      WHILE NEXTPOINT <> PERIMLPOINT DO BEGIN
        INTERIORPOINT := NEXTPOINT;
        NEXTPOINT := NEXTPOINT.NEXT;
        (* DISPOSE(INTERIORPOINT); *)
      END END;
    IF TRACEDIAGNOSTICS = ON THEN WRITELN('DISPOSE DBUG 4');
    IF TRACEPERIMS = ON THEN WRITEPERIMS(TTY,REPLACBLOB);
    (* CONNECT THE RIGHT END POINTS OF THE ABSORBED AND REPLACING SECTIONS *)
  )

```

```

IF PERINLPPOINT = PERINRPOINT THEN
  ADDRIGHTPERINPOINT(ABSORBBLOB, PERIN, NEXT, LINENUM, PERINRPOINT, COL)
ELSE BEGIN
  PERINRPOINT, NEXT := ABSORBBLOB, PERIN, NEXT, RIGHT;
  ABSORBBLOB, PERIN, NEXT, RIGHT := PERIN, RIGHT;
  PERIN, RIGHT := ABSORBBLOB, PERIN, RIGHT;
  IF TRACEDIAGNOSTICS = ON THEN WRITELN('DISPOSE DEBUG 4B');
  (* MERGE PERIMETER SECTION RINGS OF THE ABSORBED BLOB AND THE REPLACING BLOB *)
  OLDPERIN := ABSORBBLOB, PERIN;
  IF (OLDPERIN = OLDPERIN, NEXT) AND (OLDPERIN = OLDPERIN, PREV)
  THEN BEGIN (* DISPOSE(OLDPERIN) *) END (*NSP*)
  ELSE
    IF REPLACBLOB <> ABSORBBLOB
    THEN BEGIN
      (* INSERT PERIMETER SECTIONS OF ABSORBED BLOB BEFORE PERIN *)
      IF TRACEDIAGNOSTICS = ON THEN WRITELN('DISPOSE DEBUG 7');
      OLDPERIN, PREV, NEXT := PERIN;
      OLDPERIN, NEXT, PREV := PERIN, PREV;
      PERIN, PREV, NEXT := OLDPERIN, NEXT;
      PERIN, PREV := OLDPERIN, PREV;
      PERIN := OLDPERIN, NEXT;
      (* DISPOSE(OLDPERIN) *) (*NSP*)
    END
    ELSE BEGIN
      (* ADD PERIMETER LIST TO THE BLOB DESCRIPTOR OF THE TERMINATING BLOB (WHICH
      HAS THE BACKGROUND COLOR IF REPLACING BLOB = ABSORBED BLOB) *)
      (* INSERT PERIMETER SECTIONS OF ABSORBED BLOB AFTER PERIN *)
      IF TRACEDIAGNOSTICS = ON THEN WRITELN('DISPOSE DEBUG 8');
      PERIN, NEXT, PREV := OLDPERIN, PREV;
      OLDPERIN, PREV, NEXT := PERIN, NEXT;
      PERIN := PERIN, NEXT;
      TERMBLOB, PERIN := OLDPERIN;
      OLDPERIN, RIGHT := PERINLPPOINT;
      OLDPERIN, PREV := OLDPERIN;
      OLDPERIN, NEXT := OLDPERIN;
      RECORDBLOB(TERMBLOB) END;
      IF REPLACBLOB <> ABSORBBLOB THEN AREA := AREA + ABSORBBLOB, AREA;
    END; (* IF COLOR <> BKGD *)
    END; (* WITH REPLACBLOB *)
    IF ABSORBBLOB <> REPLACBLOB THEN BEGIN
      (* NOTE -- THE PARENT IS NOT COMPUTED CORRECTLY *)
      IF TRACEPARENT THEN WRITELN('LINE ', LINENUM, 3, ' ABSORBBLOB=',
      ABSORBBLOB, COMP, 3, ' REPLACBLOB=', REPLACBLOB, COMP);
      (* CHANGE ALL INSTANCES OF THE ABSORBED COMPONENT IN THE ACTIVE LINE TO
      THE REPLACING COMPONENT *)
      SEG := ACTIVELINE;
      WHILE SEG <> NIL DO BEGIN
        IF SEG, BLOB = ABSORBBLOB THEN SEG, BLOB := REPLACBLOB;
        WITH SEG, BLOB DO IF PARENT = ABSORBBLOB THEN BEGIN;
          PARENT := REPLACBLOB;
          IF TRACEPARENT THEN WRITELN('REPLACING PARENT IN SEGMENT FOR BLOB #', COMP, 4); END;
        SEG := SEG, NEXT END;
        (* CHANGE ALL INSTANCES OF THE ABSORBED COMPONENT NUMBER IN THE
        LIST OF BLOBS DONE TO THE REPLACING COMPONENT NUMBER *)
        END; (* IF ABSORBBLOB <> REPLACBLOB *)
        (* DISPOSE(ABSORBBLOB) *) (*NSP*)
        IF TRACEDIAGNOSTICS = ON THEN WRITELN('DISPOSE DEBUG 6');
        (* DELETE THE CURRENT SEGMENT AND THE SEGMENT FOLLOWING THE CURRENT ONE FROM THE
        ACTIVE LINE; SET THE CURRENT SEGMENT POINTER TO THE FIRST SEGMENT AFTER THE
        DELETED ONES *)
        PREVSEG, NEXT := CURRSEG, NEXT, NEXT;
        (* DISPOSE(CURRSEG, NEXT) *) (*NSP*)
        (* DISPOSE(CURRSEG) *) (*NSP*)
        CURRSEG := PREVSEG, NEXT;
        IF TRACEACTIVELINESEGMENTS = ON THEN TYPEACTIVELINESEGMENTS;
        IF TRACEDBLOBS = ON THEN TYPEBLOBS(TTY);
        IF TRACEPERINS = ON THEN TYPEPERINS(TTY);
        END; (* DELETESegment *)

```

```

(*SP* NEW PAGE *)
PROCEDURE UPDATEPERIM(VAR PERIM:PTRPERIMSECTION; CURRSTARTCOL,CURRENDCOL,
NEWSTARTCOL,NEWENDCOL:INTEGER);

(* FOR CASE 3 ONLY *)
VAR COL: INTEGER;
BEGIN
  (* LEFT SIDE OF PERIMETER SECTION *)
  IF CURRSTARTCOL+1 < NEWSTARTCOL THEN
    FOR COL := CURRSTARTCOL+1 TO NEWSTARTCOL-1 DO
      ADDLEFTPERIMPOINT(PERIM,LINENUM-1,COL);

  IF CURRSTARTCOL-1 > NEWSTARTCOL THEN
    FOR COL := CURRSTARTCOL-1 DOWNTO NEWSTARTCOL+1 DO
      ADDLEFTPERIMPOINT(PERIM,LINENUM,COL);

  ADDLEFTPERIMPOINT(PERIM,LINENUM,NEWSTARTCOL);
  PERIM := PERIM.NEXT;  (* ADVANCE PERIMETER SECTION POINTER *)

  (* RIGHT SIDE *)
  IF CURRENDCOL > NEWENDCOL + 1 THEN
    FOR COL := CURRENDCOL-1 DOWNTO NEWENDCOL+1 DO
      ADDRIGHTPERIMPOINT(PERIM,LINENUM-1,COL);

  IF CURRENDCOL < NEWENDCOL - 1 THEN
    FOR COL := CURRENDCOL+1 TO NEWENDCOL-1 DO
      ADDRIGHTPERIMPOINT(PERIM,LINENUM,COL);

  ADDRIGHTPERIMPOINT(PERIM,LINENUM,NEWENDCOL);
  END;  (* UPDATEPERIM *)

(*SP* NEW PAGE *)
PROCEDURE CASE3PROCESSING(VAR CURRSEG:SEGPTR; STARTCOL,ENDCOL:INTEGER);
VAR NEWAREA,XLINEAVE: REAL;
BEGIN
  IF TRACEDIAGNOSTICS = ON THEN WRITELN('BEGIN CASE 3 PROCESSING');
  WITH CURRSEG.BLOB DO
    IF COMP <> 0 THEN AREA := AREA + ENDCOL - STARTCOL + 1;

    IF (CURRSEG.BLOB.COLOR <> BKOND) AND (CURRSEG.BLOB.YMIN <> LINENUM) THEN
      UPDATEPERIM(CURRSEG.BLOB.PERIM,CURRSEG.STARTCOL,CURRSEG.ENDCOL,STARTCOL,ENDCOL);

  IF TRACEBLOBS = ON THEN WRITERBLOB(TTY,CURRSEG.BLOB);
  IF TRACEPERIMS = ON THEN WRITEPERIMS(TTY,CURRSEG.BLOB);
  END;  (* CASE3PROCESSING *)

(*SP* NEW PAGE *)
PROCEDURE PROCESSLINE;
VAR ONEZERO: INTEGER;
BEGIN
  IF LINENUM = 1 THEN BEGIN
    NEW(BLOB);  (* CREATE A BLOB DESCRIPTOR FOR THE BACKGROUND *)
    WITH BLOB DO BEGIN
      COLOR := BKOND;
      COMP := 0;
      PERIM := NIL;
      AREA := 0.0;
      XMEAN := 0.0;
      YMEAN := 0.0;
      XMIN := 0;
      XMAX := 0;
      YMIN := 0;
      YMAX := 0;
      PARENT := NIL;
      NEXT := NIL;  END;

    (* INITIALIZE THE ACTIVE LINE *)
    NEW(ACTIVELINE);  (* THE FIRST SEGMENT CORRESPONDS TO THE BACKGROUND
                      WHICH IS TO THE LEFT OF THE FIRST COLUMN *)
    ACTIVELINE.STARTCOL := MININT+1;
    ACTIVELINE.ENDCOL := MAXINT-2;
    ACTIVELINE.BLOB := BLOB;
    NEW(CURRSEG);  (* THIS SEGMENT DESCRIPTOR CORRESPONDS TO THE BACKGROUND
                   WHICH IS TO THE RIGHT OF THE LAST COLUMN *)
    ACTIVELINE.NEXT := CURRSEG;
    CURRSEG.STARTCOL := MAXINT-1;
    CURRSEG.ENDCOL := MAXINT-1;
    CURRSEG.BLOB := BLOB;
    CURRSEG.NEXT := NIL;
  END;  (* IF LINENUM = 1 *)

```

```
IF LINENUM > NSKIP THEN BEGIN
```

```
  CURRSEG := ACTIVELINE; (* INITIALIZE THE CURRENT SEGMENT POINTER *)
  NEWSSEG := NEWLINE; (* INITIALIZE THE NEW SEGMENT POINTER *)
```

```
  REPEAT
```

```
    WITH NEWSSEG DO BEGIN
```

```
      IF TRACEDIAGNOSTICS = ON THEN BEGIN
```

```
        WRITELN('CURRSEG: STARTCOL = ', CURRSEG: STARTCOL, ' CURRSEG: ENDCOL = ', CURRSEG: ENDCOL);
```

```
        WRITELN('NEWSSEG: STARTCOL = ', STARTCOL, ' NEWSSEG: ENDCOL = ', ENDCOL) END;
```

```
        IF CURRSEG: BLOB: COLOR = INK THEN ONEZERO := 1
```

```
          ELSE ONEZERO := 0;
```

```
        WHILE STARTCOL > CURRSEG: ENDCOL + ONEZERO DO DELETESegment; (* CASE 1 *)
```

```
        IF TRACEDIAGNOSTICS = ON THEN WRITELN('DEBUG PROCESSLINE 1');
```

```
        IF ENDCOL < CURRSEG: STARTCOL - ONEZERO THEN INSERTSegment(STARTCOL, ENDCOL); (* CASE 2 *)
```

```
        CASE3PROCESSING(CURRSEG, STARTCOL, ENDCOL);
```

```
        CURRSEG: STARTCOL := STARTCOL;
```

```
        CURRSEG: ENDCOL := ENDCOL;
```

```
      END; (* WITH NEWSSEG *)
```

```
      PREVSEG := CURRSEG; (* ADVANCE THE PREVIOUS SEGMENT POINTER *)
```

```
      CURRSEG := CURRSEG: NEXT; (* ADVANCE THE CURRENT SEGMENT POINTER *)
```

```
      NEWSSEG := NEWSSEG: NEXT (* ADVANCE THE NEW SEGMENT POINTER *)
```

```
    UNTIL NEWSSEG = LASTNEWSSEG: NEXT;
```

```
    IF TRACEACTIVELINESEGMENTS = ON THEN TYPEACTIVELINESEGMENTS;
```

```
    IF CURRSEG = NIL THEN BEGIN WRITELN(TTY, 'ERROR: CURRSEG=NIL'); PAUSE END;
```

```
    WHILE CURRSEG: NEXT <> NIL DO DELETESegment; (* PROCESS ALL SEGMENTS REMAINING  
      IN THE ACTIVE LINE EXCEPT THE  
      LAST SEGMENT *)
```

```
  END; (* IF LINENUM > NSKIP *)
```

```
END; (* PROCESSLINE *)
```

```
(*P* NEW PAGE *)
```

```
BEGIN
```

```
(* MAIN PROGRAM *)
```

```
  HANDONLY := TRUE; BLOBWRITTEN := FALSE;
```

```
  WRITELN('TYPE INPUT IMAGE NAME:');
```

```
  READLN; READ(INPUTNAME);
```

```
(*NSP*)
```

```
  RESET(INPUTIMAGE, INPUTNAME);
```

```
(*NSP*)
```

```
  REWRITE(DATAPFILE, DUMMY.DAT);
```

```
(*NSP*)
```

```
  WRITELN('TYPE OUTPUT PERIMETER FILE NAME:');
```

```
  READLN; READ(PERINNAME);
```

```
(*NSP*)
```

```
  REWRITE(PERINFILE, PERINNAME);
```

```
(*NSP*)
```

```
  WRITELN('TYPE THRESHOLD FOR BLOB AREA:');
```

```
  READLN; READ(AREATH); AREATH := IAREATH;
```

```
  WRITELN('PERIMETER LIST COMPACTED EVERY X LINES; TYPE X:');
```

```
  READLN; READ(LINEGROUP);
```

```
  TRACEPARENT := FALSE;
```

```
  WRITELN('TYPE # OF LINES TO BE SKIPPED AT THE TOP OF THE IMAGE:');
```

```
  READLN; READ(NSKIP);
```

```
  RECURS := FALSE;
```

```
  WRITELN('TYPE 'I' FOR INTERACTIVE DEBUGGING:');
```

```
  WRITELN(' OR 'U' FOR UNINTERRUPTED PROCESSING:');
```

```
  READLN; READ(CMD);
```

```
  IF CMD = 'I' THEN INTERACTIVE := TRUE
```

```
    ELSE INTERACTIVE := FALSE;
```

```
  WRITELN('TYPE 'T' TO TYPE OUT LINE NUMBERS:');
```

```
  READLN; READ(ONOFF);
```

```
  IF ONOFF = 'T' THEN TYPELINENUMS := TRUE;
```

```
  LINENUM := 0; NEWCOMPNUM := 0; BLOSDONE := NIL; RECYCLEDPTR := NIL;
```

```
  CHARTOT := 0;
```

```
  NEWV(ARRAYP[1]);
```

```
  FOR I := 1 TO 10 DO BEGIN
```

```
    NEWV(ARRAYP[I+1]);
```

```
    ARRAYP[I+1]: NEXT := ARRAYP[I+1]; END;
```

```
  ARRAYP[11]: NEXT := NIL;
```

```
  IF NOT INTERACTIVE THEN NLREAD := 1;
```

```
  TRACEBLOBS := OFF; TRACEPERINS := OFF; TRACERUNLENGTHS := OFF;
```

```
  TRACEACTIVELINESEGMENTS := OFF; TRACEDIAGNOSTICS := OFF;
```

```
  CMDSET := ['L', 'B', 'P', 'A', 'R', 'D', 'S'];
```

```
  REPEAT
```

```
    IF INTERACTIVE THEN BEGIN
```

```
      WRITELN('TYPE COMMAND: 'L', 'B', 'P', 'R', 'A', 'D', 'OR 'S');;
```

```
      READLN; READ(CMD);
```

```
      IF (CMD = LINEIN) THEN BEGIN
```

```
        WRITELN('TYPE NUMBER OF LINES TO BE READ:');
```

```
        READLN; READ(NLREAD) END
```

```
      ELSE NLREAD := 1
```

```
        END
```

```
    ELSE CMD := LINEIN;
```

```

IF CMD = LINEIN THEN FOR NTIMES := 1 TO NLRD DO BEGIN
  LINENUM := LINENUM + 1;
  IF TYPELINUMS THEN WRITELN('LINE ',LINENUM:3);
  GETLINE;
IF (LINENUM = 1) OR (LINENUM > NSKIP) THEN BEGIN
  IF LINENUM MOD LINEGROUP = 0 THEN PACKPERIMS;
  SETLINE;
  CREATERUNLENGTHS;
  PROCESSLINE;
END;
END;

ELSE IF (CMD <> STOP) AND (CMD IN CMDSET) THEN BEGIN
  READ(ONOFF);
  CASE CMD OF
    DIAGNOSTICS: TRACEDIAGNOSTICS := ONOFF;
    LISTBLOBS: TRACEBLOBS := ONOFF;
    LISTPERIMS: TRACEPERIMS := ONOFF;
    LISTRUNLENGTHS: TRACERUNLENGTHS := ONOFF;
    LISTACTIVELINESEGMENTS: TRACEACTIVELINESEGMENTS := ONOFF;
  END; (* CASE *)
END; (* IF *)

UNTIL (CMD = STOP) OR (LINENUM = NLINES) OR BLOBWRITTEN;
LINENUM := LINENUM + 1;
SETLINE; (* PROCESS A LINE WITH NO RUNLENGTHS TO *)
PROCESSLINE; (* ENSURE COMPLETION OF BLOB PROCESSING *)
WRITELN('LINEBYLIN -- ALL DONE');
END.

```

PROGRAM SCAN

C - Program to read unformatted sequential data files of hand data
 C - that were generated with prog LINEBYLIN on EIKONIX scans.
 C - and extract positions in space of the fingers and the corresponding
 C - interdigital spaces.

```

C-----
1  FORMAT (I4,I4,I4)
2  FORMAT ('/' Type in filename of input data file ')
4  FORMAT (Q,34A1)
5  FORMAT (I4)
6  FORMAT (' FINGER #' INCLUDES INTERDIGITAL SPACES - 1 = LITTLE
  * FINGER')
7  FORMAT (/IX,'FINGER #' 3X 'ARRAY POSITION' 3X 'XVAL' 5X 'YVAL'
  * 5X 'LENGTH' 4X 'WIDTH' 2X 'RATIO L/W' 3X 'SLOPE' 4X 'DIST')
8  FORMAT (2X,12,10X,15,8X,F8.2,X,F8.2,X,F8.2,X,F8.2,
  * X,F8.2,X,F8.2,X,F8.2,2X,F8.3)
9  FORMAT (/8' Length of scan window (12=6mm) = ')
10 FORMAT (I2)
11 FORMAT (/IX,' Length of scan window = ',I4,' (12 = 6 mm)')
12 FORMAT (/8' Angle for fingers = ')
13 FORMAT (F6.0)
14 FORMAT (/8' Angle for thumb = ')
15 FORMAT (/8' Finger angle = ',F6.0,' Thumb angle = ',F6.0)
16 FORMAT (////IX,' Input file ',I4A1,2X,' Output file ',I4A1)
17 FORMAT (2X,I6,6(2X,F8.2))
19 FORMAT (/ At Thumb Crotch: Handwidth = ',F6.2,2X,' Perim = ',
  * F13.2,2X,' Area = ',F13.2,2X,' Ratio (P**2/A) = ',F6.3)
20 FORMAT (/8' Type 1 to have data printed out at end of test ')
21 FORMAT (/8' Type 1 to mark measure points on palm plot ')
22 FORMAT (7(X,I5))
23 FORMAT (/ Entire Hand: Perim = ',F13.2,2X,' Area = ',F13.2,2X,
  * ' Ratio (P**2/A) = ',F6.3)
24 FORMAT (' Type 1 if you want to change parameters ')
25 FORMAT (3I3,X,7F10.3,/, (10X,7F10.3))
26 FORMAT (' Type numeric codes for SUBJECT ID, SESSION, TRIAL ')
27 FORMAT (I4,I4,I4)
28 FORMAT (/ Type in filename for output data file ')
29 FORMAT (/ Type DIST and R criterion for curvature anal')
30 FORMAT (F8.2,F8.3)
31 FORMAT (/ Type number (odd) of points to skipped at start')
C-----

```

```

DIMENSION END(2), MID(2), HEAD(2), TEND(2)
DIMENSION FNGTIP(18,2), FNGLEN(18), FNGWID(18), VALUE(30)
DIMENSION SLOPE(10), ZYER0(10), LINE(30)
DIMENSION TANGLE(100), DLENG(100), CURVE(10)
INTEGER IDATA(6000), FINGER, FNGNUM(18), IWIDTH(18), IDIN(6000)
LOGICAL*1 IFILE(34), IFILE2(34)
C-----

```

```

ANGLE1 = 90.      !Default criterion angle shift for fingertips
ANGLE2 = 90.      !Default criterion angle shift for thumb
IDIS = 18         !Default dist between points
DISTC = 40.0      !Starting dist from fingertip for curve
CRIT = 0.98       !Criterion fit for curvature fa.
ISTART = 201      !Skip these first points
IFLAG = 0
IF = 0
TYPE 2
ACCEPT 4, IS1, (IFILE(J), J=1,IS1)
TYPE 28
ACCEPT 4, IS2, (IFILE2(J), J=1,IS2)

```

```

TYPE 26
ACCEPT 27, IAUTH, ISESS, ITRIAL
TYPE 28
ACCEPT 5, IFLAG
TYPE 21
ACCEPT 5, NPLTF
TYPE 24
ACCEPT 5, IF
IF (IF .EQ. 0) GO TO 49
TYPE 9
ACCEPT 5, IDIS
IDIS = IDIS/2
TYPE 12
ACCEPT 13, ANGLE1
TYPE 14
ACCEPT 13, ANGLE2
TYPE 29
ACCEPT 30, DISTC, CRIT
TYPE 31
ACCEPT 5, ISTART
40 CALL ASSIGN(1, IFILE, IS1)
DO 50 K=1, 32000, 2
50 READ (1, 1, END=51, ERR=51) IDIN(K), IDIN(K+1)
STOP ' ? NO EOF ?'
51 IF (IDIN(K) .NE. 0) GO TO 52
K = K - 2 IDISregard overrun points
GO TO 51
52 NPTS = K
CALL CLOSE(1)
CALL SORT(IDATA, IDIN, NPTS+1)

C - Angle algorithm
C - Initialize counters - find angle as move through data, constantly
C - updating the average angle. Move through data with a window
C - of size IDIS, in steps of IDIS/2. Points avegd. with triangle fn.
C - When find a large shift in the average angle (ANGLE for fingers,
C - ANGLE2 for thumb) store location of tip, reinitialize
C - average angle. Store the location of the tip in the
C - data array, as well as X and Y positions in space.
C - Calculate from these both length and width of fingers
END(2) = AVE(IDATA, ISTART+1)
END(1) = AVE(IDATA, ISTART)
MID(2) = AVE(IDATA, IDIS+ISTART+1)
MID(1) = AVE(IDATA, IDIS+ISTART)
HEAD(2) = AVE(IDATA, 2*IDIS+ISTART+1)
HEAD(1) = AVE(IDATA, 2*IDIS+ISTART)
ANGLE = ANGLE1
FINGER = 1
ICNT = 0
N = 1
ALPHA = FALPHA(END, HEAD)
SUMALP = ALPHA
AVEALP = ALPHA
DO 200 I=3*IDIS+ISTART, NPTS, IDIS
END(1) = MID(1)
END(2) = MID(2)
MID(1) = HEAD(1)
MID(2) = HEAD(2)
HEAD(2) = AVE(IDATA, I+1)
HEAD(1) = AVE(IDATA, I)
IF (I .LT. N-1) GO TO 200
ALPHA = FALPHA(END, HEAD)
IF (ABS(ALPHA-AVEALP) .LE. 100.) GO TO 70
IF (AVEALP .GT. 0.0) ALPHA = ALPHA + 360.
IF (AVEALP .LT. 0.0) ALPHA = ALPHA - 360.
70 DALPHA = ABS(ALPHA - AVEALP)
IF (FINGER .LE. 4) PRINT 17, I, END(2), END(1), HEAD(2), HEAD(1),
C *ALPHA, AVEALP
IF (ICNT .LT. 6) GO TO 99 !Move at least 10cm before test
IF (DALPHA .GT. ANGLE) GO TO 100
99 N = N + 1
ICNT = ICNT + 1
SUMALP = SUMALP + ALPHA
AVEALP = SUMALP/N
GO TO 200

C - Found a finger
C
100 N = 1
ICNT = 0
SUMALP = ALPHA
AVEALP = ALPHA
IF (FINGER .EQ. 0) ANGLE = ANGLE2
IF (FINGER .GT. 7) GO TO 105
110 C - For fingers fine tune the location of the tip
C - by drawing a line through the center of the finger and find the point
C - of intersection with the tip.
L = IDELTS(IDATA, I-IDIS-1, 52., -2)
M = IDELTS(IDATA, I-IDIS-1, 52., 2)
J = IDELTS(IDATA, I-IDIS-1, 72., -2)
K = IDELTS(IDATA, I-IDIS-1, 72., 2)

```

```

LINE(FINGER*4-3)=J-1
LINE(FINGER*4-2)=L-1
LINE(FINGER*4-1)=M-1
LINE(FINGER*4)=K-1
XAVE1 = AVE(IDATA,L)
XAVE2 = AVE(IDATA,M)
YAVE1 = AVE(IDATA,L-1)
YAVE2 = AVE(IDATA,M-1)
XLM = (XAVE1 + XAVE2)/2
YLM = (YAVE1 + YAVE2)/2
XAVE1 = AVE(IDATA,J)
XAVE2 = AVE(IDATA,K)
YAVE1 = AVE(IDATA,J-1)
YAVE2 = AVE(IDATA,K-1)
XJK = (XAVE1 + XAVE2)/2
YJK = (YAVE1 + YAVE2)/2
SLOPE(FINGER) = (YLM - YJK)/(XLM - XJK)
YZERO(FINGER) = YJK - SLOPE(FINGER)*XJK
D = 2.
L1 = L
150 L1 = L1 + 2
APRIME = FLOAT(IDATA(L1-1)) - SLOPE(FINGER) * FLOAT(IDATA(L1))
IF (ABS(YZERO(FINGER) - APRIME) .LT. D) GO TO 160
IF (L1 .LT. M) GO TO 150
L1 = L
D = 2*D
GO TO 150
160 TYPE 13, D
FNGNUM(FINGER) = L1-1
FNGTIP(FINGER,2) = IDATA(L1)
FNGTIP(FINGER,1) = IDATA(L1-1)
GO TO 190
165 FNGNUM(FINGER) = 1-IDIS
FNGTIP(FINGER,2) = MID(2)
FNGTIP(FINGER,1) = MID(1)
190 FINGER = FINGER + 1
IF (FINGER .EQ. 10) GO TO 201 !Found all fingers
200 CONTINUE
201 IDIS = IDIS*2

C - Calculate length of finger
DO 510 I=1,9,2
J = 1
IF (I .GT. 6) J = -1
C - DELTAX = FNGTIP(I,2) - FNGTIP(I+J,2)
C - DELTAY = FNGTIP(I,1) - FNGTIP(I+J,1)
C - FNGLEN(I) = SQRT(DELTAX**2 + DELTAY**2)
FNGLEN(I) = DSTNCE(IDATA,FNGNUM(I)+1,FNGNUM(I+J)+1)
510 CONTINUE
TYPE 10,1

C
C - Calculate width of fingers at a distance DIST back from tip.
DIST = 92.
DO 521 I = 1,9,2
L = IDELTS(IDATA,FNGNUM(I)+1,DIST,-2)
M = IDELTS(IDATA,FNGNUM(I)+1,DIST,2)
C - DELTAX = FLOAT(IDATA(L)) - IDATA(M)
C - DELTAY = FLOAT(IDATA(L-1)) - IDATA(M-1)
C - FNGWID(I) = SQRT(DELTAX**2 + DELTAY**2)
FNGWID(I) = DSTNCE(IDATA,L,M)
DIST = 123.
IF (I .EQ. 7) DIST = 92.
IF (I .EQ. 9) I = 10
IWIDTH(I*2) = L-1
IWIDTH(I*3) = M-1
521 CONTINUE
TYPE 10,2

C
C - Calculate curvature of fingertips from + to - DIST from tip.
C - First find delta angle for tangents around the tips, then fit
C - the angle function with a straight line and store slope and fit.
C
DIST = DISTC
DO 530 I=1,9,2
525 CALL TANANG(IDATA,TANGLE,DLENG,DIST,FNGNUM(I)+1,6,12,NP)
CALL LINFIT(DLENG,TANGLE,NP,SLOPE2,R)
IF (ABS(N) .GT. CRIT) GO TO 529
DIST = DIST - 2.0
IF (DIST .LT. 4.0) GO TO 529
GO TO 525
529 CURVE(I) = SLOPE2
CURVE(I*1) = DIST
DIST = DISTC
TYPE 531, R
531 FORMAT (F8.3)
530 CONTINUE
TYPE 10,3
C

```

```

C - Calculate Handwidth, perimeter and area 9&11 on in from the tips of
C - the little and index fingers, respectively.
  L = IDELTS(IDATA,FNGNUM(1)+1,185.,-2)
  M = IDELTS(IDATA,FNGNUM(7)+1,226.,-2)
C -
  DELTAX = FLOAT(IDATA(L) - IDATA(M))
C -
  DELTAY = FLOAT(IDATA(L-1) - IDATA(M-1))
C -
  HNDWID = SORT(DELTAX**2 + DELTAY**2)
  HNDWID = DSTNCE(IDATA,L,M)
  CALL ARPRIN(IDATA,L,M,AREA,PERIM)
  FNGAR = AREA
  FNGPRM = PERIM
  IWIDTH(2) = L-1
  IWIDTH(11) = M-1
  TYPE 10,4

C - Find area and perimeter -- terminate hand from point 10cm
C - proximal to the tip of the thumb to a point 13cm proximal to the
C - tip of the little finger.
  L = IDELTS(IDATA,FNGNUM(1)+1,267.,-2)
  M = IDELTS(IDATA,FNGNUM(9)+1,265.,-2)
  IWIDTH(1) = L-1
  IWIDTH(14) = M-1
  CALL ARPRIN(IDATA,L,M,AREA,PERIM)
  TYPE 10,5

C - Print info and write feature values to file. Features written in order:
C - length, width, ratio(L/W),handwidth,finger perimeter and area,
C - ratio (finger A/P), hand perimeter and area, ratio (A/P).
  RATIO = ((FNGPRM/20.5)**2)/(FNGAR/420.)
  RATIO2 = ((PERIM/20.5)**2)/(AREA/420.)
  IF (IFLAG.NE.1) GO TO 551
  PRINT 22,(IWIDTH(1), I=1,7)
  PRINT 22,(IWIDTH(1), I=8,14)
  PRINT 16,(IFILE(1), I=1,14),(IFILE2(1), I=1,14)
  PRINT 15, ANGLE1,ANGLE2
  PRINT 11, IDIS
  PRINT 6
  PRINT 7
  DO 550 I=1,FINGER-1,2
550  PRINT 8, (I+1)/2,FNGNUM(I),FNGTIP(I,2),FNGTIP(I,1),
  *FNGLEN(I)/20.5,FNGWID(I)/20.5,FNGLEN(I)/FNGWID(I),
  *CURVE(I),CURVE(I+1)
  PRINT 19, HNDWID/20.5,FNGPRM/20.5,FNGAR/420.,RATIO
  PRINT 23, PERIM/20.5,AREA/420.,RATIO2
551  CALL ASSIGN (1,IFILE2,IS2)
  DO 600 I=1,5
    VALUE(I) = FNGLEN(I*2-1)/20.5
    VALUE(I+5) = FNGWID(I*2-1)/20.5
    VALUE(I+10) = FNGLEN(I*2-1)/FNGWID(I*2-1)
600  CONTINUE
  VALUE(16) = HNDWID/20.5
  VALUE(17) = FNGPRM/20.5
  VALUE(18) = FNGAR/420.
  VALUE(19) = RATIO
  VALUE(17) = PERIM/20.5
  VALUE(18) = AREA/420.
  VALUE(19) = RATIO2
  DO 601 I=1,9,2
    VALUE(I+19) = CURVE(I)
    VALUE(I+20) = CURVE(I+1)
601  CONTINUE
  NUMPEA = 29
  WRITE (1,25) (IAUTH,ISESS,ITRIAL,(VALUE(J), J=1,NUMPEA))

C - Plot hand perimeter numbering points for finger tips and interdial
C - spaces.
  IYOFST = 600
  IXOFST = 100
  J = 1
  CALL INITT(960)
  CALL NOVARS(IXOFST+IDATA(2),IYOFST-IDATA(1))
  DO 700 I=3,NPTS,2
    CALL DRVARS(IXOFST+IDATA(I+1),IYOFST-IDATA(I))
    IF (NPLTF.NE.1) GO TO 710
    IF (I.NE. IWIDTH(J)) GO TO 700
    GO TO 711
  C - IF (I.NE. LINE(J)) GO TO 700
  710 IF (I.NE. FNGNUM(J)) GO TO 700
  711 ENCODE(2,10,J)J
  CALL AOUTST(2,J)
  J = J + 1
  700 CONTINUE
C - Plot lines that intercept finger tips
  GO TO 1002
  DO 1000 K=1,7
    CALL NOVARS(IXOFST,IYOFST+YZERO(K))
    DO 1001 I=100,900,100
      IY = IFIX(YZERO(K) + SLOPE(K)*FLOAT(I))
      CALL DRVARS(IXOFST+I,IYOFST+IY)
1001  CONTINUE
1000  CONTINUE
1002  CALL FINITT(0,700)
  STOP
  END

```

```

FUNCTION FALPHA(END,HEAD)
C - Function to calculate the angle of this segment of the perimeter
C - using X and Y of the beginning and end of the segment
C - Angles go from 0 to 180 then -180 to 0
  DIMENSION END(1),HEAD(1)
  DATA RADIAN/57.296/,PI/3.1416/
  DELTAX = HEAD(2) - END(2)
  DELTAY = HEAD(1) - END(1)
  IF (DELTAX .NE. 0) GO TO 5
  ALPHA = PI/2. * RADIAN
  IF (DELTAY .LT. 0) ALPHA = ALPHA * -1.0
  GO TO 100
5  ALPHA = ATAN2(DELTAY,DELTAX) * RADIAN
100 FALPHA = ALPHA
  RETURN
  END

```

```

FUNCTION IMOVX(IX,MOVX,I,IDATA,LARGER,INDIRECT)
C - Function to move a distance MOVX on perimeter from point IX in
C - direction INDIRECT (+ for forward move, - for backward). LARGER is
C - a flag to tell whether the new X(Y) is going to be larger or smaller
C - than IX. Returns (I) for the new X(Y) point in the data list.
  DIMENSION IDATA(6000)
  J = I
  L = 2
  IXPRIN = IX + MOVX
  IF (INDIRECT .LT. 0) L = -2
  IF (LARGER .LT. 0) IXPRIN = IX - MOVX
  IF (LARGER .GT. 0) GO TO 50
20  J = J + L
  IF (IDATA(J) .LE. IXPRIN) GO TO 100
  GO TO 20
50  J = J + L
  IF (IDATA(J) .GT. IXPRIN) GO TO 100
  GO TO 50
100 IMOVX = J
  RETURN
  END

```

```

FUNCTION IDELTS(IDATA,L,DIST,J)
C - Function to find the location of the X,Y pair in the array IDATA that is
C - the distance DIST from the X,Y pair pointed to by L. The direction
C - and increment of search through the array is determined by J.
  DIMENSION IDATA(6000)
  FORMAT (X,'ERROR IN IDELTS')
  FORMAT (X,'IDELTS: D = ',F6.0)
  D = 2.0
  I = L
20  DELTAS = 0
  X1 = FLOAT(IDATA(I))
  Y1 = FLOAT(IDATA(I-1))
  I = I + J
  X2 = X1
  Y2 = Y1
  X1 = FLOAT(IDATA(I))
  Y1 = FLOAT(IDATA(I-1))
  DELTAX = X2 - X1
  DELTAY = Y2 - Y1
  DELTAS = DELTAS + SORT(DELTAX**2 + DELTAY**2)
  IF (ABS(DIST - DELTAS) .LT. D) GO TO 100
  IF (D .LT. 24.) GO TO 25
  TYPE 1
  STOP
25  IF (I .GT. 1) GO TO 30
  D = 2*D
  GO TO 10
30  IF (I .LT. 6000) GO TO 20
  D = 2*D
  GO TO 10
100 IDELTS = I
  RETURN
  END

```

```

FUNCTION DSTNCE(IDATA,L,N)
C - Function to calculate the distance between two points, IDATA(L)
C - and IDATA(N) using 5 point triangular averaging.
  DIMENSION IDATA(6000)
  XAVEL = AVE(IDATA,L)
  YAVEL = AVE(IDATA,L-1)
  XAVEN = AVE(IDATA,N)
  YAVEN = AVE(IDATA,N-1)
  DELTAX = XAVEL - XAVEN
  DELTAY = YAVEL - YAVEN
  DSTNCE = SORT(DELTAX**2 + DELTAY**2)
  RETURN
  END

```

FUNCTION AVE(IDATA,I)
 C - Function to do a five point triangular average and return value as
 C - AVE.

```

  DIMENSION IDATA(6000)
  PART1 = FLOAT(IDATA(I-4))*0.1
  PART2 = FLOAT(IDATA(I-2))*0.2
  PART3 = FLOAT(IDATA(I))*0.4
  PART4 = FLOAT(IDATA(I+2))*0.2
  PART5 = FLOAT(IDATA(I+4))*0.1
  AVE = PART1+PART2+PART3+PART4+PART5
  RETURN
  END

```

FUNCTION IXGET(IX,J,IDATA)
 C - Function to find a point with an X(Y) equivalent to another point (IX)
 C - J is the starting point for the search through the array.
 C - If J>10 the search will be backwards through the array.
 C - Returns (I) for the new point.

```

  DIMENSION IDATA(6000)
  N = 2
  L = 2
  IF (J .GT. 10) L = -2
  I = J
  I = I + L
  IF (ABS(IDATA(I) - IX) .LT. N) GO TO 100
  IF (I .GT. 1) GO TO 30
  N = 2*N
  GO TO 10
  30 IF (I .LT. 6000) GO TO 20
  N = 2*N
  GO TO 10
  100 IXGET = I
  RETURN
  END

```

SUBROUTINE ANPRIN(IDATA,L,N,AREA,PERIM)
 C - Routine to calculate area and perimeter of a portion of the hand with
 C - boundaries IDATA(L) to IDATA(N).

```

  DIMENSION IDATA(6000)
  AREA = 0.0
  PERIM = 0.0
  X2 = FLOAT(IDATA(L))
  Y2 = FLOAT(IDATA(L-1))
  DO 20 I=L-2,N,2
    X1 = X2
    Y1 = Y2
    X2 = FLOAT(IDATA(I))
    Y2 = FLOAT(IDATA(I-1))
    DELTAX = X2 - X1
    DELTAY = Y2 - Y1
    PERIM = PERIM + SQRT(DELTAX**2 + DELTAY**2)
    AREA = AREA + DELTAX * ((Y1+Y2)/2.0)
  20 CONTINUE
  DELTAX = FLOAT(IDATA(L) - IDATA(N))
  AREA = AREA + DELTAX * ((FLOAT(IDATA(N-1))+IDATA(L-1))/2.0)
  RETURN
  END

```

SUBROUTINE SORT(IDATA,IDIN,NPTS)
 C - Routine to sort data so that it always starts at the lower wrist.
 C -

```

  DIMENSION IDATA(6000),IDIN(6000)
  DO 10 I=NPTS,2,-2
    IF(IDIN(I) .EQ. 1) GO TO 100
  100 IF((NPTS-I) .GT. 10) GO TO 200
  DO 110 I=1,NPTS
    IDATA(I) = IDIN(I)
  110 RETURN
  L = 0
  200 DO 210 K=1-1,NPTS      (Start with Y
    L = L + 1
    IDATA(L) = IDIN(K)
  210 CONTINUE
  DO 220 K=1,I-2
    L = L + 1
    IDATA(L) = IDIN(K)
  220 CONTINUE
  RETURN
  END

```

```

      SUBROUTINE TANANG(IDATA,TANGLE,DLENG,DIST,
      *JSTART,JSTEP,JRANGE,NP)
C - Routine to calculate the angle function around a section of the
C - perimeter. The section is - to + DIST from JSTART. The length of
C - the perimeter for each tangent is JRANGE, and tangents are calculated
C - for each section in data intervals JSTEP. NP is the number of angles
C - calculated. Angles are stored in TANGLE and the corresponding
C - length is DLENG.
      DIMENSION IDATA(6000),TANGLE(100),DLENG(100)
      DATA PI/3.1416/,RADIAN/57.296/
      L = IDELTS(IDATA,JSTART,DIST,-2)
      M = IDELTS(IDATA,JSTART,DIST,2)
      NP = 0
      TLENG = 0.
      DO 100 I=L,M,JSTEP
        NP = NP + 1
        DX = AVE(IDATA,I+JRANGE) - AVE(IDATA,I)
        DY = AVE(IDATA,I+JRANGE-1) - AVE(IDATA,I-1)
        DX = FLOAT(IDATA(I+JRANGE) - IDATA(I))
        DY = FLOAT(IDATA(I+JRANGE-1) - IDATA(I-1))
        TLENG = (SQRT(DX**2 + DY**2))/2.0 + TLENG
        DLENG(NP) = TLENG
        IF (DX .NE. 0.) GO TO 20
        ANGLE = PI/2. * RADIAN
        IF (DY .LT. 0.0) ANGLE = ANGLE * -1.0
        GO TO 30
        ANGLE = ATAN2(DY,DX) * RADIAN
        IF (NP .EQ. 1) ANGLE1 = ANGLE
        IF (ABS(ANGLE-ANGLE1) .LE. 180.) GO TO 40
        IF (ANGLE1 .GT. 0.0) ANGLE = ANGLE + 360.
        IF (ANGLE1 .LT. 0.0) ANGLE = ANGLE - 360.
        TANGLE(NP) = ANGLE
        ANGLE1 = ANGLE
      100 CONTINUE
      RETURN
      END

```

```

      SUBROUTINE LINFIT(DLENG,TANGLE,NP,B,R)
C - Routine to give a least squares fit to data with a line Y=A*Bx.
C - X      array of data for independent variable
C - Y      array of data for dependent variable
C - NP     number of pairs of data points
C - A      Y intercept
C - SIGMAA standard deviation of A
C - B      slope
C - SIGMAB standard deviation of B
C - R      linear correlation coefficient
      DIMENSION TANGLE(100),DLENG(100)
      SUM = FLOAT(NP)
      SUMX = 0.
      SUMY = 0.
      SUMX2 = 0.
      SUMY2 = 0.
      SUMXY = 0.
      DO 100 I=1,NP
        X1 = DLENG(I)
        Y1 = TANGLE(I)
        SUMX = SUMX + X1
        SUMY = SUMY + Y1
        SUMX2 = SUMX2 + X1**2
        SUMY2 = SUMY2 + Y1**2
        SUMXY = SUMXY + X1*Y1
      100 CONTINUE
C - Calculate coefficients and standard deviations
      DELTA = SUM * SUMX2 - SUMX**2
      A = (SUMX2*SUMY - SUMX*SUMXY)/DELTA
      B = (SUMXY*SUM - SUMX*SUMY)/DELTA
      C = NP - 2
      VARNCE = (SUMY2*A**2 + SUM*B**2 + SUMX2 - 2*
      1(A*SUMY + B*SUMXY - A*B*SUMX))/C
      SIGMAA = SQRT(VARNCE*SUMX2/DELTA)
      SIGMAB = SQRT(VARNCE*SUM /DELTA)
      R = (SUM*SUMXY - SUMX*SUMY)/SQRT(DELTA*(SUM*SUMY2 - SUMY**2))
      RETURN
      END

```

Appendix B
DOCUMENTATION OF LINEBYLINE

Gregory K. Myers
SRI International

I INTRODUCTION

This algorithm is taken from "Image Processing Algorithms for Industrial Vision," a report by Gerry Agin of SRI International. The algorithm is referred to as "connectivity analysis." It segments a binary image into "blobs" (connected regions) of the same "color" (gray level). Only one pass is made through the image, and only one line is accessed at a time (hence the name "LINEBYLINE"). Features of each blob are computed, such as the area, center of gravity, bounding rectangle, and perimeter points. This document explains the concepts used in LINEBYLINE and provides an example of processing.

II RUN-LENGTH CODING

The first step in processing is the run-length coding of the binary image. Each line of the binary image is converted into a series of run-length segments. A "1" denotes the color of "object" and "0" denotes the color of "background."

To illustrate how run-length coding is done, let us consider the following 8 × 8 binary image:

Column number:		1	2	3	4	5	6	7	8
		b	b	b	b	b	b	b	b
Row number:	1	b	0	0	0	1	1	1	0
	2	b	1	1	0	0	1	1	0
	3	b	0	1	0	1	1	0	1
	4	b	0	1	1	1	1	0	0
	5	b	0	1	1	1	0	0	0
	6	b	0	1	0	1	1	0	1
	7	b	0	1	0	0	1	1	0
	8	b	0	0	0	0	0	1	0
		b	b	b	b	b	b	b	b

(Note: "b" denotes an implicit "0" beyond the margins of the image.)

The algorithm for run-length coding examines each pixel of the row in turn and records the column number of any element that differs from its predecessor. A 0 is assumed to precede the first element and to follow the last one. For this reason, if the last column of some particular row contains a 1, a transition will be recorded in column 9. In general, there will be twice as many column numbers recorded as there are segments of contiguous 1s in the row.

The result of run-length coding our example is as follows:

Row 1:	Col. 4,7
Row 2:	Col. 1,3,5,8
Row 3:	Col. 2,3,4,6,7,9
Row 4:	Col. 2,6,8,9
Row 5:	Col. 2,5,8,9
Row 6:	Col. 2,3,4,6,7,9
Row 7:	Col. 2,3,5,8
Row 8:	Col. 6,7

Later in this paper, we will refer to the run-length segments that make up a line or row. These are contiguous sequences of pixels of either color (0 or 1). For example, row 1 has exactly three run-length segments. The first segment is all 0s and extends from minus infinity up to (but not including) column 4. The second segment is 1s runs from column 4 to (but not including) column 7. The third and last segment consists of 0s from column 7 to plus infinity. Because every row must start and end with 0, there is always an odd number of run-length segments in any line.

The starting and ending column numbers of each run-length segment are stored in a linked list for later processing by the connectivity-analysis procedures. When the connectivity analysis for all the run-length segments in a line has been completed, the run-length segments are deleted. The creation of run-length segments from a binary image line could be easily performed in hardware.

III INTRODUCTION TO CONNECTIVITY ANALYSIS

The purpose of connectivity analysis is to separate a binary image into connected components. We call these connected components blobs. A hole is a special case of a blob entirely surrounded by another blob of the contrasting color. When connectivity analysis is applied to the 8×8 example presented in the previous section, the result contains three components: the background (component 0), an "object" (component 1), and a hole (component 2). The three components are shown below.

0 0 0 + + + 0 0	. . . 1 1 1 + + + . .
+ + 0 0 + + + 0	1 1 . . 1 1 1 .	+ + . . + + + .
0 + 0 + + . + +	. 1 . 1 1 . 1 1	. + . + + 2 + +
0 + + + + . . +	. 1 1 1 1 . . 1	. + + + + 2 2 +
0 + + + . . . +	. 1 1 1 . . . 1	. + + + 2 2 2 +
0 + 0 + + . + +	. 1 . 1 1 . 1 1	. + . + + 2 + +
0 + 0 0 + + + 0	. 1 . . 1 1 1 .	. + . . + + + .
0 0 0 0 0 + 0 0 1 + . .

All pixels in the background are assigned the same region number, even though in the 8×8 figure the background appears to consist of four separate regions. This is because we assume the image is embedded in an infinite field of 0s. Any region of 0s that touches the margin of the picture will therefore be classified as background.

The algorithm extracts and identifies connected components, computes useful feature information, and requires buffering of no more than one line of data. This section describes the general procedure for connectivity analysis. Section IV gives some details that are skipped in this section for the purpose of explaining the process more easily.

Connectivity analysis, as implemented here, makes use of a data structure, which we call the active line, to keep track of the processing. The active line consist of

- A linked list of segment descriptors. Each segment descriptor contain the starting and ending column numbers of a run-length segment, the component number to which the segment belongs, and

a pointer to the next segment descriptor. A dummy segment descriptor gives the ending column number of the final run-length segment.

- A pointer to the current segment being processed.

Before processing any data, the active line is initialized to contain a single segment descriptor, representing the background. Before processing any row of run-length data, the pointer to the current segment pointer must be initialized to point to the first segment. As we start to process our example image, the state of affairs is thus:

Image so far: b b b b b b b b b b

Active line:

Start column	-infinity	+infinity
Component number	0	dummy

Current segment: |

In the first row of example data, transitions were found in columns 4 and 7. This implies that the row consists of three run-length segments: 0s from -infinity to (but not including) column 4, 1s from column 4 to (but not including) column 7, and 0s from column 7 to +infinity. After processing the first row we will want the data structure to represent those three segments.

One procedure will examine each run-length segment in turn. Each segment will take its turn as the new segment, represented by its starting and ending column numbers. The new segment must be matched against the partially completed analysis embodied in the current line. In particular, we must determine whether the new segment overlaps the current segment, that is, the segment of the active line pointed to by the current segment pointer. Three cases are possible:

Case 1:

The two segments do not overlap because the new segment is to the right of the current one (that is, the starting column number of the new segment is larger than the ending column number of the current segment).

Current segment: XXXXXXXX
New segment: XXXXXXXX

Case 2:

The two segments do not overlap because the new segment is to the left of the current one (that is, the ending column number of the new segment is smaller than the starting column number of the current segment).

Current segment: XXXXXXXX
New segment: XXXXXXXX

Case 3:

Neither Case 1 nor Case 2 obtain, and the segments overlap.

Current segment: XXXXXXXXXXXX
New segment: XXXXXXXXXXXX

Different actions must be taken in each of the three cases. We will explain each case as it occurs in the course of analyzing our example image.

The first segment of the new row goes from $-\infty$ to 4. The current segment pointer points to the first segment in the active line, which goes from $-\infty$ to $+\infty$. When these two segments are compared, we find that Case 3 applies: the two segments overlap.

Action on Case 3:

Copy the start column number from the new segment to the current one. Then advance the current segment pointer to the next segment in the active line.

Copying the starting number of the new segment ($-\infty$) to the current segment results in no change in the data structure, because $-\infty$ was there to begin with. The current segment pointer is advanced to the next segment descriptor in the active-line list. After processing the first segment in the first row, the active-line data structure looks like this:

```

Image so far:      b b b b b b b b b b
New segment:      b 0 0 0

Active line:
    Start          -inf      +inf
    Component      0        dummy

Current segment:      |

```

We now analyze the next segment. This new segment runs from columns 4 up to (but not including) column 7. The current segment pointer now points to the dummy segment descriptor, which starts at +infinity. Therefore Case 2 applies: The new segment does not match any existing segment in the active line. Room must be made in the data structure for it.

Action on Case 2:

Insert two new segments descriptors (call them A and B) in the active line before the current segment. Let the current segment pointer point to A. Copy the ending column number of the new segment to the start column number of B. Choose an unused number for a new component and let it be the component number for A. Copy the component number from the segment preceding segment A, to B. Now proceed as in Case 3.

The situation after processing the second segment in the first row is thus:

```

Image so far:      b b b b b b b b b b
                  b 0 0 0

New segment:      1 1 1

Active line:
    Start          (A)      (B)
    Component      -inf      4      7      +inf
                  0        1      0      dummy

Current segment:      |

```

Two new segment descriptors were inserted in the active line. Descriptor A represents part of a newly discovered blob in the image; it is assigned component number 1. Descriptor B extends the background down the right side of the new blob.

Descriptor B takes its starting column number from the ending column number of the new segment (7) and its component number from the segment before A (0). Having created a new pair of segment descriptors, we can proceed to match the new segment to segment A by performing the Case 3 action. The starting column number of the new segment (4) is copied to descriptor A, and the current segment pointer is advanced to point to segment B.

The third (and last) segment in the first row runs from column 7 to +infinity. This will be matched against the current segment, which also runs from 7 to +infinity. This is another Case 3 overlap, and the data structure after processing that new segment will be:

```

Image so far:      b b b b b b b b b
                   b 0 0 0 1 1 1
New segment:                               0 0 b
Active line:
  Start      -inf      4      7      +inf
  Component   0        1      0      dummy
Current segment:                               |

```

This concludes processing of the first row.

The second row has transitions at 1, 3, 5, and 8, or five segments to be processed. The current segment pointer is reset to point to the first active segment.

The first segment of this row runs from -infinity to 1. When it is matched to the first segment in the active line, a Case 3 overlap is discovered. The appropriate action (extending the segment representing the background) will be performed.

The next segment runs from column 1 to (but not including) column 3. When it is matched against the current segment starting at column 4, Case 2 applies. The newly discovered blob is assigned component number 2, producing the following situation after processing the second segment:

Image so far: b b b b b b b b b
 b 0 0 0 1 1 1 0 0 b
 b
 New segment: 2 2
 Active line: (A) (B)
 Start -inf 1 3 4 7 +inf
 Component 0 2 0 1 0 dummy
 Current segment: |

The last three segments on this second row all produce Case 3 overlaps when they are matched against the active line. At the conclusion of processing the second row, the following data structure obtains:

Image so far: b b b b b b b b b
 b 0 0 0 1 1 1 0 0 b
 b 2 2 0 0 1 1 1
 New segment: 0 b
 Active line: -inf 1 3 5 8 +inf
 Start -inf 1 3 5 8 +inf
 Component 0 2 0 1 0 dummy
 Current segment: |

Row 3 has transitions at columns 2, 3, 4, 6, 7, and 9. Matching the seven segments of this row will result in four Case 3s, a Case 2, and two more Case 3s. After processing the third row the following will be the state of the data structure:

Image so far: b b b b b b b b b
 b 0 0 0 1 1 1 0 0 b
 b 2 2 0 0 1 1 1 0 b
 b 0 2 0 1 1 3 1 1 b
 Active line: -inf 2 3 4 6 7 9 +inf
 Start -inf 2 3 4 6 7 9 +inf
 Component 0 2 0 1 3 1 0 dummy
 Current segment: |

In the fourth row there are transitions at columns 2, 6, 8, and 9, for a total of five segments. The first two of these result in Case 3 overlaps. But when we match the third segment of the fifth row (which runs from column 6 to 8) with the next segment of the active line (running from column 3 to column 4) we discover that Case 1 holds. Here is the situation as we discover the fact:

```

Image so far:      b b b b b b b b b
                   b 0 0 0 1 1 1 0 0 b
                   b 2 2 0 0 1 1 1 0 b
                   b 0 2<0>1 1(3)1 1 b
                   b 0(2 2 2 2)

New segment:                               [0 0]

Active line:      (A)      (B)
Start      -inf      2      3      4      6      7      9      +inf
Component    0      2      0      1      3      1      0      dummy

Current segment:      |

```

The current segment in the active line, marked by "< >" above, is not matched by any segment in the new row. The new segment, marked by "[]" must eventually match the segment denoted by "()". We must delete the unmatched segment "< >" from the active line. Furthermore, the previous segment, marked by "()", forms a "bridge" between components 1 and 2. We must merge the two components.

Action on Case 1:

Consider the segment in the active line before the current one (call it "A") and the segment after the current one (call it "B"). If the component numbers of A and B are different, then merge the two components by changing all instances of B's component number in the active line to A's component number. Delete the current segment and segment B from the active line. Let the current segment pointer point to the segment after B. Now go back to the beginning of the entire matching procedure, matching the new segment against the updated active line.

By this procedure, we merge components 1 and 2. The merging changes the component numbers of the fourth and sixth segment descriptors in the active line from 1 to 2. The current segment and segment B get deleted, and the current segment pointer is advanced to point to the segment

starting at column 6. After we have taken these steps, the data structure looks as follows:

```

Image so far:*      b b b b b b b b b b
                   b 0 0 0 2 2 2 0 0 b
                   b 2 2 0 0 2 2 2 0 b
                   b 0 2 0 2 2 3 2 2 b
New segment:        b 0 2 2 2 2 3 3

Active line:        (A)
                   Start  -inf  2   6   7   9   +inf
                   Component 0   2   3   2   0   dummy

Current segment:    |

```

Now the new segment may be matched anew against the active line and current segment. This time around, the match is Case 3, as are the remainder of matches in this fourth row. After finishing this row, the state of the data structure is as shown below:

```

Image so far:      b b b b b b b b b b
                   b 0 0 0 2 2 2 0 0 b
                   b 2 2 0 0 2 2 2 0 b
                   b 0 2 0 2 2 3 2 2 b
                   b 0 2 2 2 2 3 3 2 b

Active line:
                   Start  -inf  2   6   8   9   +inf
                   Component 0   2   3   2   0   dummy

Current segment:    |

```

In the fifth row, there are five segments with transitions at columns 2, 5, 8, and 9. Only Case 3 overlaps occur.

*The "image so far" is not part of the data structure; it only is presented as an aid in visualization of the connectivity-analysis process. We have changed all 1s in the image to 2s to indicate, conceptually, the merging of the two components.

The sixth row has transitions at columns 2, 3, 4, 6, 7, and 9. The third segment (from columns 3 to 4) will generate a Case 2 match, the others will all be Case 3. The new component created by the Case 2 match is assigned number 4. Here is what the data structure looks like after processing that row:

Image so far:

b	b	b	b	b	b	b	b	b	b
b	0	0	0	2	2	2	0	0	b
b	2	2	0	0	2	2	2	0	b
b	0	2	0	2	2	3	2	2	b
b	0	2	2	2	2	3	3	2	b
b	0	2	2	2	3	3	3	2	b
b	0	2	4	2	2	3	2	2	b

Active line:

Start	-inf	2	3	4	6	7	9	+inf
Component	0	2	4	2	3	2	0	dummy

Current segment:

In processing row 7, with transitions at 2, 3, 5, and 8 a Case 1 situation occurs. The analysis of component 3 is complete: Its segment descriptor disappears from the active-line data structure. The two segments on either side of component 3 already have the same component numbers, so no merging is necessary:

Image so far:

b	b	b	b	b	b	b	b	b	b
b	0	0	0	2	2	2	0	0	b
b	2	2	0	0	2	2	2	0	b
b	0	2	0	2	2	3	2	2	b
b	0	2	2	2	2	3	3	2	b
b	0	2	2	2	3	3	3	2	b
b	0	2	4	2	2	3	2	2	b
b	0	2	4	4	2	2	2	0	b

Active line:

Start	-inf	2	3	5	8	+inf
Component	0	2	4	2	0	dummy

Current segment:

In row 8, there are transitions at columns 6 and 7. Another Case 1 situation occurs here, merging component 4 with component 0 (the background). Here is the situation after we have processed the eighth row:

```
Image so far:      b b b b b b b b b b
                   b 0 0 0 2 2 2 0 0 b
                   b 2 2 0 0 2 2 2 0 b
                   b 0 2 0 2 2 3 2 2 b
                   b 0 2 2 2 2 3 3 2 b
                   b 0 2 2 2 3 3 3 2 b
                   b 0 2 0 2 2 3 2 2 b
                   b 0 2 0 0 2 2 2 0 b
                   b 0 0 0 0 0 2 0 0 b
```

Active line:

Start	-inf	6	7	+inf
Component	0	2	0	dummy

Current segment:

1

After the last row of image data, another row of all 0s will complete the analysis, removing component 2 from the active line.

We have engaged in a little oversimplification in order to present the basic ideas behind connectivity analysis as plainly as possible. In the next section we will fix the algorithm to operate correctly in all cases. But before reading on, turn to the beginning of this section, where we presented the three components we wish the algorithm to extract. You may verify that the results we have obtained are what was intended except that different numbers were assigned.

IV CONNECTIVITY ANALYSIS IN DETAIL

The simplified algorithm presented in the previous section is deficient. It has errors, and it doesn't compute any features of the blobs or any perimeter lists. In addition, processing of some borderline cases of overlapping segments was not adequately described. At the end of this section the complete algorithm with its amendments and corrections will be presented.

A. Handling Some Special Cases

Let us finish the processing of our test image. To complete the analysis, a last row of 0s running from $-\infty$ to $+\infty$ is added to the active-line data structure, leaving us with the following situation:

```
Image so far:      b b b b b b b b b b
                   b 0 0 0 2 2 2 0 0 b
                   b 2 2 0 0 2 2 2 0 b
                   b 0 2 0 2 2 3 2 2 b
                   b 0 2 2 2 2 3 3 2 b
                   b 0 2 2 2 3 3 3 2 b
                   b 0 2 0 2 2 3 2 2 b
                   b 0 2 0 0 2 2 2 0 b
                   b 0 0 0 0 0 2 0 0 b
New segment:       b b b b b b b b b b

Active line:
  Start      -inf      6      7      +inf
Component    0        2      0      dummy

Current segment:   |
```

We have just processed a line with nothing in it, but the active line incorrectly points to a segment there. It takes a Case 1 match between segments to delete component 2 from the data structure, but there is nothing left to process. What we must do is to check at the end of each line whether the current segment pointer points to the dummy segment at the end. If it does not, then the Case 1 processing must be performed until the current segment pointer does point to the last segment.

A second error concerns the merging of two regions when a "bridge" connects them. The rule for Case 1 states that if the two component numbers are different, then the number on the right replaces the number on the left everywhere else in the active line. However, if the component number on the right happens to be the background, the replacement would renumber the background. We wish the background to be always component number 0 and must modify the renumbering rule accordingly.

B. Blob Descriptors

The main reason for doing connectivity analysis is to obtain information about the components we find thereby. The information we have derived related to color, location, area, perimeter, and points to other components that surround, neighbor, or form holes in the component we are examining. The algorithms for deriving such information will be described later in this report. Such information is stored in a blob descriptor for each component.

We must modify our algorithm description to include the use of blob descriptors. When a new component is started as a result of Case 2 processing, a new blob descriptor must be created. When segments are removed from the active line in Case 1 processing, we must keep track of the blob descriptors that are removed. Several segment descriptors in the active-line data structure can have the same component number; hence they can point to the same blob descriptor. Therefore, when a segment descriptor is removed as the result of Case 1 processing, the algorithm must check to see if other segment descriptors in the active line refer to the same component. If this is the last reference to that component, its processing is finished.

At this point, the blob descriptor describes a connected component whose connectivity has been completely determined. Furthermore, the extraction of blob features and perimeter lists is complete. We have more than one option as to what to do with the blob descriptor. We can pass the address of the completed blob descriptor to an application-dependent subroutine that will do further processing, take some action depending on what was found, or delete the blob descriptor to make its memory space available for other blobs or other purposes. Alternatively, the address of the blob descriptor can simply be added to a list of isolated blobs in the image and further processing will be deferred until the entire picture is analyzed. In the current implementation the features and perimeter lists of the blob are written to output files, and the blob descriptor is recycled to save memory space.

C. Treatment of Diagonally Adjacent Pixels

Two pixels belong to the same component if and only if there exists a path, along adjacent elements of the same color, from one pixel to the other. When dealing with points on a rectangular grid, it is reasonable to ask whether diagonal points are adjacent. If diagonal points are to be considered adjacent, then the components of an image are said to be 8-connected, since each pixel has eight connected neighbors. If diagonal points are not considered adjacent, then the components are called 4-connected. Our connectivity algorithm, as presented so far, finds the 4-connected components of an image. If we were to analyze the image of a checkerboard, then each square of the image, black or white, would be a separate component.

It would be desirable to modify the connectivity algorithm so it has the property that every component (except the background) is entirely enclosed by one and only one component, which is of the opposite color. Then every boundary between two regions is a closed curve that separates an enclosing region from an enclosed region. The components of a scene thus analyzed may be arranged hierarchically, each component having a single superior (encloser) and zero or more inferiors (holes or enclosed regions).

A way of achieving this is to specify that all white cells (1s) are 8-connected and that all black cells (0s) are 4-connected. With such a convention, the image of a checkerboard would be analyzed as a number of individual white squares, all embedded in a single black region (the background). To implement this requires modifying the tests for overlap that determine Case 1, 2, or 3. The modification has to be such that one set of tests applies when a segment of 1s is being considered and a different set of tests applies for a segment of 0s.

D. The Connectivity Algorithm

We shall now consolidate what has gone before. The algorithm will be presented in an informal fashion. Operations such as creating and modifying lists, or obtaining storage for a descriptor are implementation details that will not be mentioned here.

First, some definitions.

- Segment descriptor is an array or a block of data that contains four items: starting and ending column numbers, a pointer to a blob descriptor, and a pointer to the next segment descriptor in the active line.
- The active line is a linked list of segment descriptors. The current segment pointer points to a segment descriptor on this list. The active segment is the segment descriptor pointed to by the current segment pointer.
- A blob descriptor is a record of data that contains at least one item: a color, which may be either 0 or 1. Additional items in a blob can be used for feature analysis, as will be described later. A component number is a number that identifies a blob.

To process an image:

- Obtain a blob descriptor to represent the background. Set the color word of the background blob to 0.
- Initialize the active line to contain two segment descriptors. The first segment descriptor should have a column number smaller than zero and a component number pointing to the background blob. The column number of the second segment descriptor should be a large positive number, a number greater than the number of columns in the image. The component number of the second descriptor is irrelevant.
- Process each row of the image, as described below.
- Finish by processing an extra row consisting of all zeros.

To process a row:

- Initialize the current segment pointer to point to the first segment descriptor in the active line.
- Obtain the run-length representation of the row (see Section II). The run-length data should start with a negative number and end with a large positive number.
- For every pair of adjacent numbers in the run-length data, in turn, call the segment-processing operation (defined below).
- While* the current segment pointer does not point to the last segment in the active line, perform the deletion operation (defined below). Repeat this step until the current segment pointer does point to the last segment.

* In this context, the word while refers to a condition to check and an action to perform. It means to repeat the action zero or more times until the condition is not true.

To process a new segment, given a starting column number and an ending column number:

- While the starting column number is greater than the ending column number of the current segment in the active line, do the deletion operation (defined below). (This is Case 1.)
- If the ending column number is less than the starting column number of the current segment, perform the insertion operation (defined below), passing on the starting and ending numbers of the segment. (This is Case 2.)
- Do feature extraction for Case 3 processing.
- Copy the starting and ending column numbers of the new segment to the starting and ending column numbers of the current segment.
- Advance the current segment pointer to point to the next segment descriptor in the active line.

To perform the insertion operation, given starting and ending column numbers:

- Obtain the component number of the segment descriptor preceding the current segment. Call that component the surrounding component.
- Obtain a new blob descriptor. Call it the new component. Set the color word of the new component to the opposite of the color of the surrounding component.
- Obtain two new segment descriptors and insert them in the active line immediately before the current segment, calling the first segment "A" and the second "B." Segment A receives the new component number and the starting column number. Segment B receives the surrounding component number and the ending column number.

To perform the deletion operation:

- Call the component number of the current segment the terminated component. Call the component number of the segment preceding the current one the left component and the component number of the segment following the current one the right component. Call the left component the replacing component and the right component the replaced component. If the right component points to the background, then call it the replacing component and the left component the replaced component; otherwise, call the left component replacing and the right component replaced.
- Do feature extraction for Case 1 processing.
- If the replacing component and the replaced component are different, then find all instances of the replaced component number in the active line, and change them to the replacing number.

- Delete the current segment and the segment following the current one from the active line. Let the current segment pointer point to the first segment after the deleted one.
- Search for instances of the terminated component number in the active line. If there are no remaining instances, call application-dependent subroutines, as appropriate, passing the address of the terminated component's blob descriptor.

V EXTRACTION OF FEATURES

The preceding sections outlined a procedure for isolating connected components (blobs) from one another in an image. The results of the analysis are a number of blob descriptor records that contain information about the blob. Each blob descriptor consists of several features or characteristics of interest.

A. Color

The simplest feature is color. The color of a blob will be 0 if the blob is black and 1 if it is white. If the color of the background is appropriately set up at initialization time, then the following will ensure that each blob descriptor subsequently created has the correct color: whenever a new blob descriptor is created (Case 2) obtain the color feature of the surrounding blob and store the opposite color in the appropriate place in the new blob descriptor.

B. Parent, Child, and Sibling

An important class of features describes inclusion relationships among blobs. Because we have been careful in the way connections are made between diagonally adjacent pixels, we can guarantee that every component (except the background) has one and only one surrounding component blob. We call any blob's surrounder its parent, any holes (or blobs enclosed by this one) children, and other blobs enclosed by the same parent siblings.

At blob creation time (Case 2) a pointer to the surrounding blob descriptor can be stored as the new blob's parent feature. The child

feature of the new blob is set to 0. We must also record the fact that the parent blob has an additional child. There is only a single entry in the blob descriptor for the child feature. The child feature of any blob points to the most recently created surrounded blob; the sibling feature of that most recently created child points to the next most recently created child, and so forth. The procedure to follow in Case 2 processing is as follows: Copy the child feature of the surrounding blob to the sibling feature of the new blob. Set the child feature of the surrounding blob to point to the new blob.

Case 1 processing frequently causes two blobs to be merged into one. Topological considerations guarantee that the two blobs to be merged will have the same color feature and the same parent. But to preserve the integrity of the parent-child-sibling links, several actions must be performed. In each of the children of the replaced blob, the parent feature must be changed to point to the replacing blob. All the children of the replaced blob must be concatenated onto the sibling list of the children of the replacing blob. The pointer to the replaced blob itself must be removed from the sibling list that includes both the replaced and the replacing blobs. Finally, if the common parent of the two merged blobs points to the replacing blob as its child feature, that child feature must be replaced by the contents of the sibling feature of the replaced blob.

A great deal of bookkeeping overhead is eliminated by recording only parent relationships. The children and siblings can be regenerated from the list of parent relationships. In the current implementation of LINEBYLINE, the parent relationship is not computed because it is not used by subsequent programs.

C. The Bounding Rectangle

The bounding rectangle of a blob is specified by the minimum and maximum values of x and y over the blob. It is easily extracted as follows: when a blob is created (Case 2), copy the current value of y (which is constant across an entire line of input) to the new blob's ymin feature. At the same time place the starting column number (of the

contrasting run-length segment) in the new blob's xmin feature and the ending column number in the blob's ymin feature.

For each run-length segment added to a blob (Case 3), set the xmin feature to the smaller of (1) the starting column number and (2) the previous value of the xmin feature. Do the analogous operation for the xmax feature and the ending column number. The ymax feature is set to the current line number.

When two blobs are merged as a result of Case 1 processing, the ymin, xmin, and xmax features of the replacing blob must take the extreme values of the two blobs to be merged.

D. Area and Center of Gravity

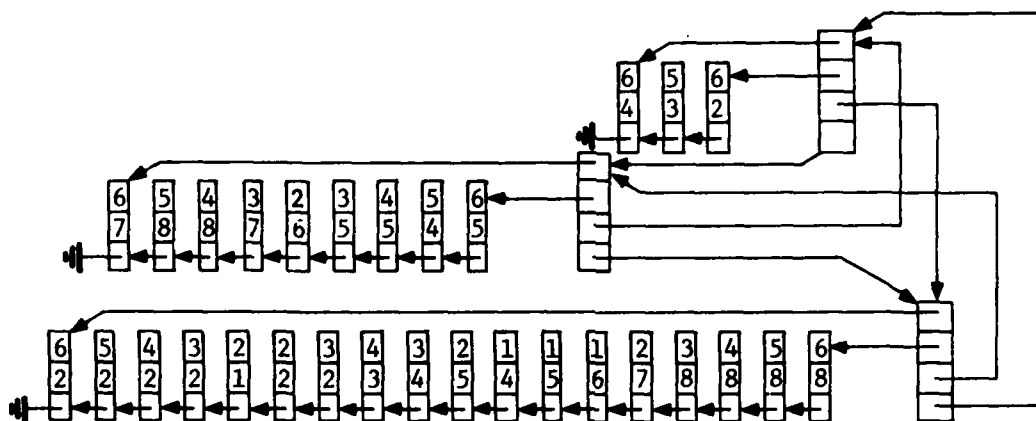
The area of a blob is set to 0 when the blob descriptor is created. When a new line segment of the blob is processed, the number of pixels in the segment is added to the area of the blob. When two blobs are merged, their areas are added together.

The center of gravity (CG) is recomputed after each new line segment is added to the blob. The updated CG is a weighted average of the CG of the new line and the CG of the previously processed portion of the blob. The weights are proportional to the areas of the new line segment and the previously processed portion, respectively. When two blobs are merged, a weighted average of the two CG's is computed in a similar manner as above.

E. Perimeter Lists

Associated with each inked blob is a set of perimeter points. Because each line of the image is processed sequentially, the perimeter points are obtained in a somewhat haphazard manner. During processing, the perimeter of a simple convex blob has two uncompleted ends: One at the starting column of the active line segment (the left end), and one at the ending column of the active line segment (the right end). The perimeter points are stored as a singly linked list called a perimeter section. The right endpoint is at the beginning of the list, and the

left endpoint is at the end of the list. When there is more than one active line segment corresponding to the same blob, one perimeter section is associated with each active line segment. The perimeter sections are arranged as a doubly linked ring. This structure is illustrated below, using component 2 in the previous example after six lines of processing.



The structure of the perimeter section records and perimeter point records in the illustration above are shown here:

left
right
previous section
next section

Perimeter section record

line
column
next point

Perimeter point record

When each new line of pixels is processed, the lists of perimeter points are modified according to which case applies. In all cases perimeter lists are compiled for inked blobs only.

Case 3 requires the simplest processing. When a new line segment is added to an inked blob, the left end of the associated perimeter section and the right end of the next perimeter section are extended. An example is shown below:

Image so far:

```

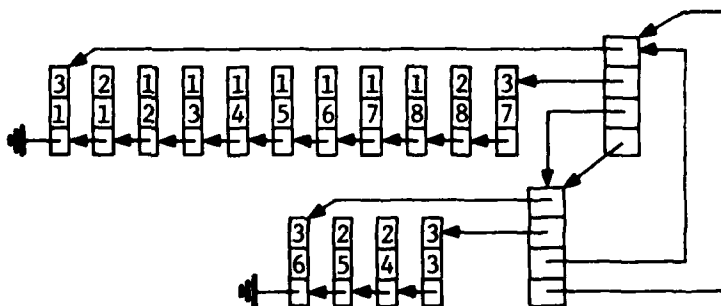
b b b b b b b b b
b 0 1 1 1 1 1 1 1 b
b 1 1 1 1 1 1 1 1 b
b 1 1 1 0 0 1 1 0 b
b

```

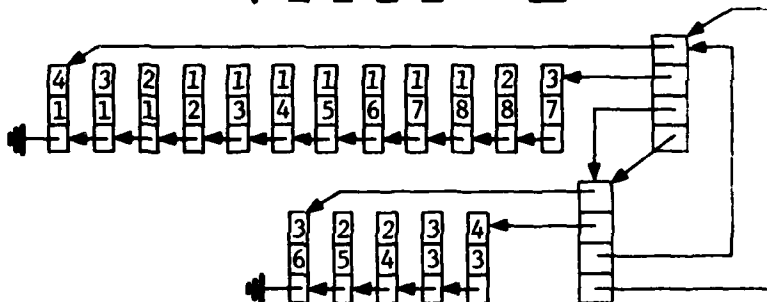
New segment:

[1 1 1]

Perimeter lists
before new seg-
ment is added:



Perimeter lists
after new seg-
ment is added:



For Case 2 (insertion of a new segment) there are two subcases. Subcase A occurs if the new segment has the color ink; otherwise Subcase B occurs. In Subcase A a new perimeter section for the new blob is created and will contain all of the points in the new segment. An example is shown below.

Image so far:

```

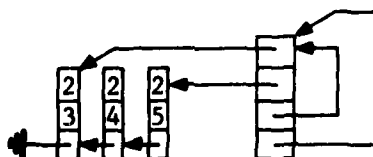
b b b b b b b b b
b 0 0 0 0 0 0 0 0 b
b 0 0

```

New segment:

[1 1 1]

Perimeter list after
new segment is added:



In Subcase B the perimeter section of the surrounding segment is split into two perimeter sections. An example is shown below.

Image so far:

```

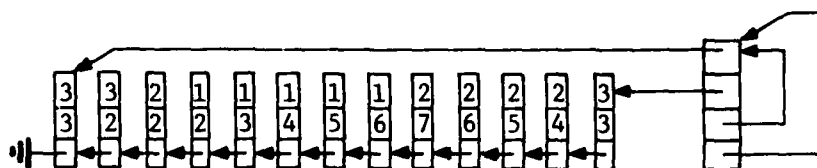
b b b b b b b b b
b 0 1 1 1 1 1 0 0 b
b 0 1 1 1 1 1 1 0 b
b 0 1 1

```

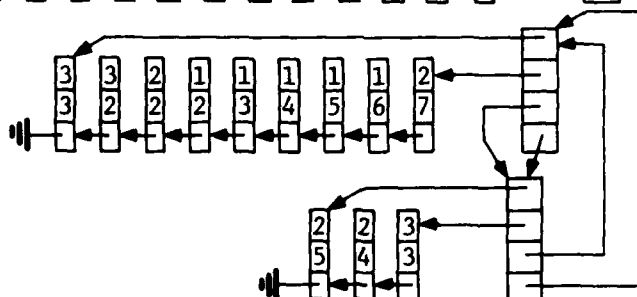
New segment:

[0 0]

Perimeter lists
before new seg-
ment is added:



Perimeter lists
after new seg-
ment is added:



For Case 1 (deletion of a segment) there are also two subcases. Subcase A occurs if the deleted segment has the color ink; otherwise Subcase B occurs. In Subcase A two adjacent perimeter sections of the terminating blob are merged. If the terminating blob has only one perimeter section, its left and right ends are joined, and processing of this blob is finished. An example is shown below.

Image so far:

```

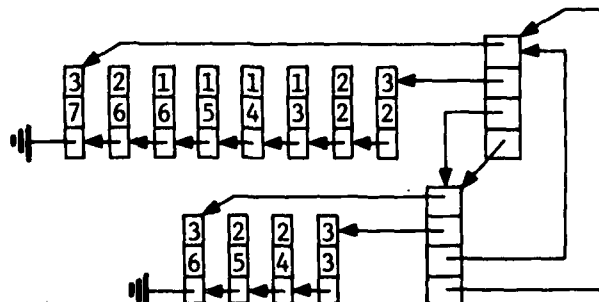
b b b b b b b b b
b 0 0 1 1 1 1 0 0 b
b 0 1 1 1 1 1 0 0 b
b

```

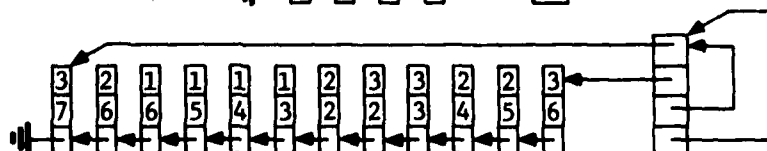
New segment:

[0 0]

Perimeter lists
before new seg-
ment is added:



Perimeter lists
after new seg-
ment is added:



In Subcase B the perimeter sections associated with the replacing and replaced components are merged. If the replacing and replaced components belong to different blobs, the two blobs will be merged and the two rings of perimeter sections will be merged into one ring. An example is shown below.

Image so far:

```

b b b b b b b b b
b 1 1 0 0 0 1 1 0 b
b 1 1 0 0 1 1 0 0 b
b

```

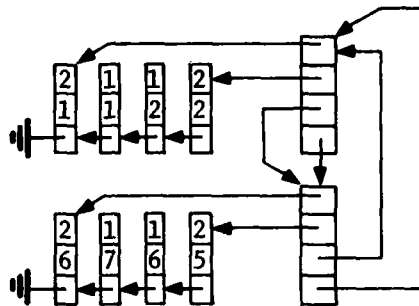
New segment:

```

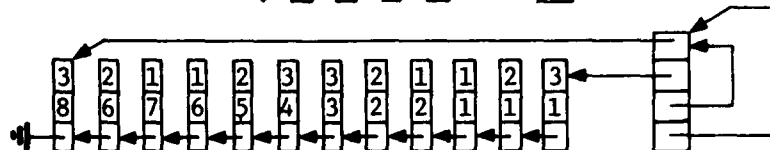
[1 1 1 1 1 1 1]

```

Perimeter lists
before new seg-
ment is added:



Perimeter lists
after new seg-
ment is added:



To save space during processing of the image, and to store the completed perimeter lists in less disk space, the perimeter lists are periodically packed. The variable LINEGROUP controls how often the lists are compressed (the words "pack" and "compress" mean the same thing). For example, when LINEGROUP = 25, compression occurs at lines 25, 50, 75, etc. Those points in the list that previously have been compressed are skipped. When a perimeter list has been completed and is ready to be written to disk, it is unpacked and then repacked if the Boolean variable FINALPACKING is true. If FINALPACKING is false, the perimeter list is only unpacked. Currently, the value of FINALPACKING is specified by the user at the beginning of LINEBYLINE.

Each unpacked perimeter point consists of two 16-bit integers with values between 1 and 512. One integer is for the line number Y and the

second integer is for the column number X. Compression is achieved by converting the perimeter information from a series of (Y,X) points to a starting (Y,X) point and a series of directions from one point to the next. The starting point has the same format as an unpacked perimeter point, except that 512 is added to the column number. Therefore, a column number greater than 512 identifies that point as the beginning of a series of compressed points. Because the perimeter is 8-connected, there are eight possible directions indicating the positions of subsequent perimeter points. They are numbered from 0 to 7 and are assigned as follows:

	Change in X			
	-1	0	1	
	-1	3	2	1
Change in Y	0	4		0
	1	5	6	7

For example, if the perimeter list of the last example above (deletion of a segment, Subcase B) were to be compressed starting with the point (3,1), the direction to the next point (2,1) would be 2; the direction from (2,1) to the next point (1,1) would be again 2; and the direction from (1,1) to the next point (1,2) would be 0.

Each of the eight directions can be represented as a 3-bit code (000 = 0, 001 = 1, 010 = 2, ..., 111 = 7). Five direction codes fill 15 bits of a 16-bit word. The 16th bit (the sign bit) is set equal to 1 to indicate that the word contains compressed perimeter points. Each set of five direction codes is stored alternately in the line and column locations of the perimeter point record. If we continue the example, the first five directions would be 2, 2, 0, 6, and 7. The would be coded and placed in a 16-bit word as:

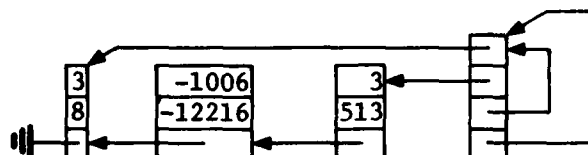
1	1	1	1	1	1	0	0	0	0	1	0	0	1	0	
16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

This word is interpreted as the two's complement integer -1756 octal, or -1006 decimal. The second five directions are 0, 1, 1, 0, and 5. Their bit representation is:

1	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0
16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

This is interpreted as -27670 octal, or -12216 decimal. Hence, the first 11 perimeter points are compressed into the space of two perimeter point records as follows: (3,513), (-1006,-12216).

Compression of groups of less than 10 points is not attempted. Therefore, during processing, the list of compressed points is interspersed with uncompressed points. In addition, there are usually a few uncompressed points at the end of the list. In our example, the 12th point, (3,8) remains uncompressed. The completely compressed list is therefore:



The unpacking process is just the reverse of the packing process. The column number of each perimeter point is examined. If it is greater than 512, the start of a series of compressed points has been detected. Subtracting 512 from the column number results in the (Y,X) coordinate of the starting point. The sign of the column number of the next perimeter point is examined. If it is less than or equal to 0, then compression of 10 points has been detected. The locations of these points are recomputed from the 3-bit direction codes that are extracted from the two 16-bit words. This process continues until a perimeter point record with a column number greater than 0 is encountered.

VI HIERARCHY OF PROCEDURES

This chart shows the hierarchy of procedures within LINEBYLINE. All of the procedures that are called by each procedure are indented and listed below.

```
LINEBYLINE
  PACKPERIMS
    PKPERIMS
      COMPRESSED
      UNPACKPERIMS
        UNPACK5
        DIRPOINT
      DELETEPERIMS
      NEWPERIM
      DIRECTION
  GETLINE
    ENDOFBLOCK
  SETLINE
  CREATERUNLENGTHS
    ADDRLENGTHS
  PROCESSLINE
    DELETESEGMENT
      ADDLEFTPERIMPOINT
      NEWPERIM
      ADDRRIGHTPERIMPOINT
      NEWPERIM
      RECORDBLOB
      WRITEPERIMS
        WRITEPOINTS
          PKPERIMS
            (see above)
          COUNTANDWRITE
          COMPRESSED
          UNPACKPERIMS
            (see above)
          WRITEPOINTS
            (see above)
          DELETEPERIMS
      WRITEBLOB
      DELETEBLOB
  INSERTSEGMENT
    NEWBLOB
    ADDRRIGHTPERIMPOINT
      (see above)
  CASE3PROCESSING
    UPDATEPERIMS
      ADDLEFTPERIMPOINT
        (see above)
      ADDRRIGHTPERIMPOINT
        (see above)
```

VII INPUT IMAGE FORMAT

The binary image that is read by LINEBYLINE is 512×512 pixels. There are 8 pixels per byte (each pixel is one bit). Therefore, one line of data occupies 64 bytes, and the entire image occupies $512 \times 64 = 32K$ bytes on disk. The data is read one byte (one character in PASCAL) at a time.

However, there is one small difference due to the PDP 11/40 version of PASCAL. The bytes are stored on disk in blocks of 512 bytes. The PASCAL I/O processor interprets a block boundary as an extra byte. To compensate for this inaccuracy, the LINEBYLINE software skips an "imaginary" byte once every 512 times (when the Boolean function ENDOFBLOCK is true). On most computer systems, this would not be necessary.

Appendix C

ANNOTATIONS FOR THE SEPARATE SPSS RESULT PACKAGE
(Submitted Separately)

Appendix C

ANNOTATIONS FOR THE SPSS RESULT PACKAGE (Submitted Separately)

The printout of an SPSS analysis contains several parts. Following is a description of each part with the page number of the listing on which that part of the printout begins. Preceding the actual output, some initial information is listed, including the control file used to run SPSS and the data that are to be read by SPSS. These data are ordered as a sequence of trials. The numbers represent: Subject code (1-30), session number, trial number, and 29 feature values.

- Page 1: SPSS description; format of the data that are to be read in.
- Page 3: A partial list of the data that were read in. Each trial is listed, but only five selected feature values are listed.
- Page 9: SPSS options used.
- Page 10: Number of trials per subject (group).
- Page 10: Feature-value means for each subject.
- Page 12: Feature-value standard deviations for each subject.
- Page 15: Wilk's lambda and univariate F-ratios for each feature. Each of these numbers is related to the discriminating power of the specific feature. For Wilk's lambda, the smaller the number, the better is the discriminating power of that feature. The opposite is true for F-ratios: The larger the F-ratio, the greater is the discriminating power of that feature.
- Page 17: Within-groups correlation matrix. This is similar to the covariance matrix, except that each feature mean is first normalized to mean = 0 and standard deviation = 1 before calculation of the covariance. A number in this matrix that approaches 1 indicates a high degree of correlation between those two features..
- Page 19: Prior probability of finding any particular subject within the population. A priori, this probability was set to be equal for each subject.

- Page 19: Twenty-nine discriminant functions were determined. The data in this chart demonstrate the declining discriminating power of each additional discriminant function.
- Page 20: The standard discriminant-function coefficients are listed. The product of all normalized feature values and the appropriate discriminant-function coefficient will be the discriminant score used to classify each subject.
- Page 23: Here start listings of both the unstandardized discriminant-function coefficients and the location in 29-dimensional space of the centroids of each group. The centroid is the mean discriminant-function score for each subject.
- Page 30: Classification results: Actual group is the subject code number; "highest group D**2" is the classification result. In all cases the group was classified correctly. "Probability $P(G/x)$ " is the probability that the sample belongs to Group G given that the discriminant measure x was observed. "Probability $P(x/G)$ " is the probability that the discriminant measure x would be observed given that the sample is from Group G. The actual discriminant scores (upon which the classification is based) are listed on the far right.
- Page 78: Territorial map of first two (most discriminating) discriminant functions. This is a 2D representation of the data. The centroids (*) are separated by numbers or letters (corresponding to that particular group, numbered 0-9 or A-Z) that define the equal probability lines between groups.



*MISSION
of
Rome Air Development Center*

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C³I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.

DAT
ILM